Version of 29 November 2010

Towards a science of open source systems

Final Report
November 2010

Prepared for the Computing Community Consortium (CCC)

Walt Scacchi, Kevin Crowston, Chris Jensen, Greg Madey, Megan Squire
Thomas Alspaugh, Les Gasser, Scott Hissam, Yuzo Kanomata, Hamid Ekbia,
Kangning Wei, Charles Schweik
and others from the *2010 FOSS Workshop on the Future of Research in Free/Open Source Software*

Newport Beach, CA
10-12 February 2010
http://foss2010.isr.uci.edu

Table of Contents

Executive Summary

We seek to establish a national program for research into the science of open source systems.

Open source systems are beginning to appear in many diverse disciplines, though perhaps the area with the highest level of activity, visibility, and impact is free/open source software (FOSS) systems. FOSS systems are being researched and developed by fast growing communities of academic and industrial practitioners in different disciplines. However, FOSS systems are much more than just source code, or software applications; they are better understood as packages of interrelated social and technical resources that interact and overlap, and that can occasionally give rise to profound consequences. This report addresses and elaborates on the nature of FOSS systems in order to identify the questions and problems that will guide research in this domain over the next five to ten years. Further, it provides a set of recommendations for action targeted to FOSS researchers, research agencies, and others involved in scientific research and technology development.

How are FOSS systems developed?, How do people working at a distance from each other build them? How does such work draw on surrounding webs of resources and socio-technical relationships? How do these systems evolve over time? These are questions of growing importance to the future of software engineering, education, innovation, science, society, and government. This report details the published research studies and the open research problems that together describe the current state of scientific knowledge about FOSS systems. Yet as FOSS systems permeate more aspects of science, technology, society, and government, we will be limited in our collective ability to explain, rationalize, predict, control, develop and transfer these systems. Consequently, we identify and recommend the research studies, research infrastructures, and other resources needed to expand the scientific knowledge we have started to produce.

This report is organized into four major sections.

The first section motivates a deliberate scientific study of FOSS systems. It discusses how FOSS systems are developed, why FOSS will help create the new scientific knowledge we now lack, and how such knowledge can be transformative as an engine of innovation in a growing number of application areas. This section details how FOSS systems and development practices are creating new research practices in many scientific domains and can contribute to resolving many outstanding issues in large-scale software engineering. FOSS systems are also changing global software and Information Technology (IT) industries. The development and deployment of FOSS

systems can further transform society and cultural practices. In short, FOSS systems and their development constitute a rich, fertile area for scientific research and technology development that will create new knowledge about software systems through computational thinking and practice.

The second section serves to elaborate on the specifics of what researchers know about FOSS systems, and what remains to be discovered. This section is organized into four sub-sections: (1) the *development processes, work practices, and project forms* that facilitate and shape FOSS systems; (2) the *collaboration processes* that govern how people who work at a distance from each other with little or no face-to-face interaction can develop complex FOSS systems together, often without formal project management regimes, budget, or schedules; (3) the surrounding web of resources and socio-technical relationships that constitute the *FOSS ecosystems* that contextualize, situate, nurture, sustain, and adapt what FOSS systems will be developed, and to what ends; (4) the *evolution of FOSS systems* over time, and how open access to FOSS system evolution, data and artifacts is giving rise to surprising results in system evolution, such as sustained exponential growth across many system generations.

The third section addresses the kinds of data, repositories, access and analysis tools, and other resources that comprise the research infrastructure needed for systematic empirical studies of FOSS systems. Such a research infrastructure will support further investigation in the areas described in Section Two: the way FOSS systems are developed, the collaboration practices in FOSS, FOSS ecosystems, and FOSS evolution. Additionally, the science of open source systems that we are pursuing is one in which open source, open access, open archiving, open distribution and dissemination are essential elements of the open science we seek to foster, practice, and model. The research instrumentation that we need to grow and expand our body of scientific knowledge depends on cultivating and supporting this diverse web of FOSS system research infrastructures.

The fourth section summarizes the broad implications of a sustained research program and significant investment in the science of FOSS systems in particular, and into open source systems more generally. These implications will be seen most readily in the domains of large-scale software development; education and learning; social and technological innovation; and science, industry, and government. Finally, the report ends with a set of nine recommendations for action that detail the ways that organizations or agencies can maximize their research investments. These recommendations are listed here to help set the stage for the remainder of this report. However, they are not simply a recapitulation of findings detailed elsewhere in the report. Instead, they are intended as recommendations for action based on an understanding of what the results from current studies of FOSS systems, processes and practices, collaboration patterns,

software ecosystems, and evolution processes imply for the advancement of scientific knowledge and broader technological development.

Recommendations:

1. Stimulate investment in projects for scientific research and/or technology development that build FOSS systems as a way to stimulate workforce development.

2. Create a new cross-cutting research program or office within the CISE Directorate that supports all aspects of FOSS systems research—FOSS development processes, work practices, and alternative project forms; collaboration in development and use of FOSS systems; FOSS ecosystems; and FOSS system evolution.

3. Stimulate the FOSS research community and others to focus research attention on the development and use of FOSS systems in other science research programs, as well as in health, energy, climate, defense, and National Engineering Challenge domains.

4. Stimulate new research into substantially under-explored areas such as Gender and FOSS, and Collaboration and Diversity in FOSSD.

5. Invest in and encourage cross-cultural studies of FOSS, especially comparative studies of FOSSD activities in non-English speaking cultures and countries.

6. Stimulate the research and development of FOSS systems for humanitarian aid and relief applications, especially those that provide opportunities for graduate, undergraduate, and secondary students to participate and contribute.

7. Stimulate existing research programs in Software Engineering, Human-Centered Computing, and Networking Technology and Systems to investigate and develop new approaches to the challenges of engineering FOSS systems and real-world systems that rely on FOSS.

8. Establish and support shared research repositories for FOSS data as part of the new research infrastructure that supports FOSS systems research.

9. Pursue development of advanced data analysis tools for examining FOSS data as part of the new FOSS systems research infrastructure.

Finally, this report should not be viewed as a definitive statement about progress in FOSS systems research to date, nor should it be seen as a list of ultimate goals for the future of research into FOSS systems. As our goal in producing this report was to increase awareness of the advances and challenges that are found in this emerging, open area of computer and information science and engineering research, we reiterate our belief that this report must also be an open source, collaborative endeavor whose

interaction with its surrounding ecosystems of scholars, enthusiasts, and critics will collectively create the meaning and content of this report. Consequently, we will make the text of the most current version of this report available on the FOSS 2010 workshop Web site (http://foss2010.isr.uci.edu), along with an archive of previously released versions (in PDF format), so that others can also contribute to the ongoing refinement of our collected knowledge of the future of research into free and open source software systems.

Part I

The Science of FOSS

Motivational Transformations

Overview

This report describes our vision for a national program for research into the science of open source systems.

Based on a community workshop, conference presentations, and supporting meetings we conducted in the winter and spring of 2010, our intention is to collectively identify the major questions that will shape future research in FOSS systems, the research resources and infrastructures needed to explore and resolve these questions, and to articulate our vision of a future in which FOSS plays an ever larger role in information technology and society. Specifically, we see that major topics of research consider how FOSS systems shape and are shaped by FOSS development processes, practices, project communities,m collaboration practices and ecosystems that situate FOSS systems; and how FOSS systems evolve. In turn, advances in the development of missing scientific knowledge in these areas will help articulate the beneficial impacts and contributions of FOSS systems to software development, education, and innovation practices. Similarly, the new research infrastructure and resources required to conduct the research that will produce this knowledge is also identified. Consequently, we begin by defining what open source systems are, why they are significant and merit research study, and why FOSS systems are the primary area to study to develop the missing knowledge needed to realize the full range of scientific and societal benefits that can follow from the widespread adoption of open source system development and use.

What are open source systems and what are FOSS systems?

Open source systems are those whose operational description and technological embodiment can be interpreted and used by both computers and people. This is most readily observable in FOSS systems in which the software source code is an operational description of how to perform a system of algorithmic processes, data manipulations, and user-computer interactions, as well as the means for instructing a computer system to perform associated computations. Open source systems can also be found to a lesser (but growing) degree in fields focusing on hardware design (open source microprocessors, do-it-yourself home-made devices), product design (open source cars, open source cameras), cultural media (film, video, music, art, online documents, Wikipedia), and, importantly, the science common in scientific research. In

some of these cases part or all of the descriptions are in computational form, but the final embodiment may be available in non-computationally executed form (e.g., a tangible manufactured artifact that, once made, cannot be easily modified, remade, and redistributed). In other cases, new derivations are continually encouraged. But FOSS systems are the open source systems receiving the greatest attention, investment, and effort to develop and deploy new technologies in ever more diverse disciplines. Moreover, they are an area in which openness and permission for new derivations through collaborative processes on the Internet have been established protocol for some time. There is much to be learned from them that can inform these activities in other, emerging domains.

We propose a new science of open source systems to understand how and why FOSS systems sometimes give rise to the hugely successful, socioeconomically beneficial, and transformational information technologies that underlie much of the Internet and Web, yet at other times fail to gain sufficient traction to engage much interest. How are FOSS systems developed and used, and how do they sometimes evolve into transformational systems and communities of practice? How do we create a new generation of transformational systems using FOSS in mission-critical problem domains like health care, cybersecurity, or global climate change?Can we even create such systems without FOSS? These are fundamental questions that we lack the scientific knowledge to answer reliably or predictably. Furthermore, they are different questions than those that motivate research into software applications, services and tools, and their development.

FOSS systems are much more than source code. The Open Source Initiative (OSI -- http://www.osi.com), the community-recognized body for reviewing and approving FOSS licenses that comply with the “open source definition” recites eleven criteria that define software that can be publicly identified and licensed as FOSS. These criteria include: free redistribution, inclusion of source code and executable/compiled code, allowance for modification and creation of derived works, maintaining the integrity of the author's source code, not discriminating against any persons or groups when providing FOSS, not discriminating against use of FOSS in a specific field of endeavor, allowance for distribution of licenses without the need for additional licenses, requiring that the license for a FOSS product must not be product specific, must not restrict other software, and must be technology neutral.

A FOSS system comes as a *socio-technical package* — a package that links social and technological resources through development processes, work practices, and project communities that continuously transform collective action into new technologies, and technologies into new kinds of work organizations [Kling and Scacchi 1982; Star and Ruhleder 1996]. What distinguishes FOSS systems packages from previous computing packages is that FOSS systems are open, accessible online, observable and

modifiable. Their redistribution, mobilization, and redeployment are encouraged with minimal restrictions. Such packages interrelate and include:

- Online information artifacts (source code, electronic bulletin boards, threaded email discussion, bug reports, etc.).
- Development processes, work practices, and project community dynamics that collectively develop and sustain continuously evolving software applications, services, and tools.
- Online information infrastructure (e.g., supporting software tools and techniques, internet servers, licenses, standard practices and tools, and project artifact repositories).
- Communities of practitioners who develop, use, or otherwise contribute to what is working well – and what isn't.
- Data sets manipulated by the FOSS system being developed or used.
- Philosophies or ideologies about resource sharing practices.
- Histories, or information legacies, of work accomplishments including bug reporting systems, individual and group blogs, etc.
- Systems of capital (social, technical, human, etc.).
- New roles, such as community manager, super reviewer or bug reporter, in which to participate and contribute to the ongoing effort.
- Intellectual property licenses that reinforce social relationships, values, and beliefs (e.g., the General Public License, GPL).
- Governance structures and other institutions guiding the collaboration, and evolving as the project evolves.

Each FOSS system, whether based in a single FOSS development (FOSSD) project, or spanning multiple ecosystems of interrelated FOSSD projects [Jensen and Scacchi 2005], is similar to and different from all others in ways that the FOSS research community is just beginning to understand. At the same time, entrepreneurs and venture capitalists are increasingly shifting their focus away from exclusive proprietary software products and towards those that intermingle proprietary and FOSS components, as a strategy to reduce time to market, improve product quality and

service opportunities, and reduce software development and marketing costs [Augustin 2010]. Furthermore, science researchers in fields outside of Computer and Information Science and Engineering (CISE) disciplines, as well as Information Technology (IT) practitioners in different corporate, governmental, and academic enterprises, are increasing their interest and investment in FOSS systems. Once again, we lack deep fundamental knowledge to explain why this is so, and why prior Computer Science research efforts in areas like software engineering have not generated this kind of knowledge or enthusiasm for emerging software systems.

Where does FOSS Research belong?

In considering a new proposal for research that advances our scientific and technical knowledge about the development, use, and evolution of FOSS systems, one may ask where does FOSS fit into CISE research programs? This is the question that can be addressed through a set of questions and answers that offer insight into how to "rethink software" [Wing, Hirsh, *et al*. 2008].

Is FOSS just another name for software that is not proprietary? Is it nothing more than a software free lunch?

If so, FOSS does not represent a fundamentally different approach to developing software. It is nothing new, and research into FOSS can be addressed using existing notions of software and software research programs already in place at the National Science Foundation (e.g., CCF Software and Hardware Foundations (SHF) Program, CNS Computer Systems Research (CSR) Program, Information and Intelligent Systems (IIS) Programs, and CISE Cross-Cutting (CC) Programs). If FOSS fits here, it fails to "...articulate new software research challenges that cannot be addressed with existing software concepts, methods and tools" [Wing, Hirsh, *et al*. 2008]. But empirical study of FOSS development projects already reveals that: (a) how FOSS is developed and evolved does not conform to, nor can it be explained by traditional "software life cycle" models; (b) a common dependence on evolving and self-referential webs of online information artifacts articulate FOSS system requirements and design; (c) a multiplicity of FOSS projects (some numbering into the hundreds) are developing alternative versions of the same software system, functions, features, or services in order to realize user preference; (d) social networks of FOSS developers and users collectively provide the critical mass for sustaining exponential growth of successful FOSS systems; and (e) the overwhelming majority of nascent FOSS projects fail to produce usable software systems. Though FOSS is software, conditions (a-e) are not readily explainable with extant software foundations and engineering practices that focus attention on programming languages, formal semantics, data abstractions, or software engineering principles. Instead, research studies of FOSS focus on the people, tools, and processes

involved in creating, integrating, updating, managing, and supporting complex FOSS systems, projects, and communities to better explain the causes and variations of how FOSS comes to be the way it is (cf. IIS in [Wing, Hirsh, *et al*. 2008]).

If the future of computing is being driven bynew technologies like massively multi-core microprocessors, cloud computing, and pervasive computing [Wing, Hirsh, et al. 2008], what role if any can FOSS play to help realize or accelerate scientific advances in these areas?

There are many competing R&D efforts underway to determine how best to program multi-core processors using alternative programming languages or data abstractions in mainstream Computer Science research. Intel and AMD have released open source software development tools for their multi-core and many-core processors. But the release of FOSS system tools by itself does not necessarily enable the formation of a community of FOSS system developers and end-users to use such tools. Instead, a quick look at the Hadoop FOSS technology and community (which focuses on one approach to programming massively parallel software application systems for cloud computing) already finds hundreds of CS researchers in academia and industry developing and using FOSS tools, development processes, work practices, and community Web sites to self-identify and organize like-minded people contributing to the Hadoop software ecosystem [Hadoop 2010]. In other words, Hadoop succeeds in part because the FOSS and its surrounding community of contributors are part of the socio-technical cyberinfrastructure of resources that are available to new Hadoop contributors, along with the parallel computing systems that might not otherwise be available to researchers. Though we generalize with caution, it appears that future advances in computing technologies like multi-core processors and pervasive computing may be better enabled by efforts that focus on cultivating, nurturing, and sustaining self-organizing, online, and decentralized communities of developers and end-users who will invest time, skill, and effort into engaging FOSS tools, development processes, work practices, and project forms open for participation and contribution to advance these technologies. This may be a more effective strategy than simply developing and giving away software tools that support new computing technologies. Once again, FOSS is best viewed not as software, but as a community-centered approach to developing, using, and evolving complex software systems.

What is the "science" of FOSS systems, and how can it contribute to advances in other scientific research endeavors?

To date, the science of FOSS systems can be found in empirical studies "...centered on the people, tools, and processes involved in creating, integrating, updating, managing, and supporting complex software,...open source software" (cf. IIS in Wing, Hirsh, *et al*. [2008]). FOSS science focuses on exploring, explaining, and modeling FOSS source code, artifacts, processes, projects, communities, and knowledge within or across

FOSS projects [Gasser and Scacchi 2008]. The science of FOSS is mostly an empirical and systematic observational endeavor giving rise to new models and theories, and in some cases the creation of new software systems, rather than one focusing on the creation or analysis of new software languages, formalisms, or abstractions. The practice of FOSS within other scientific disciplines can sometimes be found as the means for engaging in scientific observation and experimentation. At other times it can be found in the results of scientific research. FOSS development processes, work practices, and project community dynamics are being put to work in R&D projects now underway in subjects such as Economics (motivations for FOSS developers; industry competitiveness), Law (FOSS license regimes), Public Policy (impact on balance of trade, FOSS adoption by local governments), Art (open source and open media artworks), Anthropology (FOSS practices in non-Western cultures), Organization Science (end-user innovation, public-private innovation approaches), Business/Management (corporate adoption of FOSS, maintainability of FOSS), Geography (FOSS-based Geographic Information Systems), Biology (open source bioinformatics), Physics (astrophysics software, Large Hadron Collider software), E-science software (open source grid software, workflow and research provenance software), and Information Systems (understanding teamwork in FOSS development, success factors in FOSS development). However, some of these efforts have suffered from nominal understanding of FOSS, while some early Computer Science-based studies of FOSS slight or ignore the social and community aspects essential to sustained FOSS projects. Moreover, in many of these disciplines, there is emerging interest in expanding the “science” of FOSS systems into collaborative domains outside of traditional software development. This expansion has great potential for harnessing human innovation on a global scale. However, significant questions exist about how and to what degree these systems of production differ from the more traditional domains of software engineering and development.

It is clear that FOSS is reshaping the research agendas of scholars in many scientific and cultural disciplines. FOSS is emerging as a way of rethinking software as a complex web of socio-technical networks that intertwine people, online artifacts, processes, projects, communities and knowledge [Gasser and Scacchi 2008]. These socio-technical interaction networks may be able to transform research practices across disciplines, global software/IT industries, society and culture in ways driven by participatory innovation practices.

Why we need a national research program in FOSS systems

In addition to the need to develop new scientific knowledge about how FOSS systems are developed and used, there are a number of related motivations to study FOSS systems. Though FOSS systems are widely used, we believe that much of the

Computer Science research community has yet to fully recognize their potential to change the world of research and development of software-intensive systems across disciplines. Software from thousands of FOSS system projects are widely used and globally distributed, tens of thousands of FOSS system projects are up and running worldwide, and millions of end-users increasingly rely on FOSS systems. Growing numbers of research projects in physical, social, and human sciences, as well as the cultural arts, now routinely expect to develop or use FOSS systems to best meet their needs. Similarly, growing numbers of businesses and government organizations are looking to develop and use mission-critical software applications that are built with FOSS system components. We believe such investment in FOSS can be attributed to the following observations.

FOSS exists as a high-impact socio-technical phenomenon on its own right

FOSS is important not just as an alternative to traditional software engineering, but as a software process that is successful in many dimensions. We should foster and improve FOSS, and we should study it in order to describe, archive, and build it. It took open source software to create, study, modify, improve and spread the Internet and Web. FOSS systems are more than source code. FOSS is both a social movement and a technical approach for developing complex software-intensive systems, a package of resources and relationships that can produce complex socio-technical computing systems. Accordingly, we need to understand and study the ***history*** of FOSS systems.

FOSS system code and related artifacts can be accessed, studied, modified, archived, and redistributed by anyone

FOSS systems are public, unlike proprietary software. FOSS represents a low-cost running start for many kinds of innovation, as growing numbers of FOSS applications become available in different configurations. Substantial FOSS application platform stacks can be easily located, downloaded, installed, and redistributed from online sources such as Bitnami.org (http://www.bitnami.org) and PortableApps.com (http://www.portableapps.com), where these source code bases often exceed a million lines of source code. FOSS systems can serve as the bases for software engineering (SE) concepts in the context of large software systems that are open for study, modification and experimentation.

For cases where we need more openness and transparency, such as the source code for citizen voting systems and cybersecurity, FOSS represents a new and potentially better way than proprietary closed source development approaches.

The openness and transparency we see in FOSS projects and FOSS systems offers a second benefit. The public availability of data within a project and similarities in data format across projects hosted by the same FOSS distribution infrastructure (e.g., SourceForge.net) provides opportunities for longitudinal studies of many systems using common research instrumentation. Additionally, the availability of such data facilitates the repeated study of a project or system. By contrast, replication studies in software engineering are few and far between [Sjoeberg, D.I.K., *et al.* 2005].

FOSS is a technological "extremophile" in several domains

FOSS is one of the few viable examples of widely used and ongoing successful communities of practice relying on commons-based peer production to create complex systems. The discipline of Software Engineering has long established and refined practices for building software systems. These SE practices embrace the use of formalized schemes, notations, analytical methods, and automated tools that seek to provide a robust approach to construction of reliable software systems. SE practices were traditionally performed in a single location with an explicit regime for software project management, budgeting and scheduling, where software engineers were expected to be motivated by salary and technical challenges. FOSS has a different lineage. Some FOSS systems, like the Debian/GNU Linux distribution now exceed more than 500M lines of source code spanning 23,000+ software packages, which indicates it would best be considered an example of an Ultra-Large-Scale (ULS) System [Northrop, Feiler, *et al.* 2006], as well as one of the most complex human artifacts ever created — yet it is publicly available for free to anyone interested in downloading it. If traditional economic assumptions about individual motivation in performing work don't apply, what are the motivations of software developers who work without payg? What happens when volunteer developers collaborate with other FOSS developers who are paid to do their job? FOSS represents a different set of incentives from traditional forms of collaborative production. FOSS is like a "model organism" making inroads into new processes, raising questions similar to those asked of high-temperature bacteria and related life forms: How do they do that?! To what degree are they morphing and transforming into different forms of collaboration and collective action?

FOSS is a new phenomenon in software development. The FOSS community has moved ahead to address complex system issues in ways that are starting to work outside traditional SE practice. Further, some FOSS systems are now very large or ultra large systems in their own terms, and unlike other complex systems in the modern

world and global economy, FOSS systems are open, public, and amenable to close-up study, modification, and redistribution with comparatively few barriers. FOSS systems are therefore likely to become the bases for or central components of complex systems in other domains.

FOSS system development is participatory and engages active user involvement

Compared to prior software development approaches that emphasize technical system functionality (e.g., service-oriented architecture, object-orientation, computer-aided software engineering, structured programming and other disciplined software engineering methods [Boehm and Turner 2003]), FOSS development is both socially convivial and technically engaging. FOSS developers are often end-users of the software they build, so the division between developers and end-users is eliminated. This simplifies difficult software development activities like requirements specification, analysis and testing, since developers know first-hand what they want and need. Finally, FOSS projects create new ways to participate in development as a *contributor:* providing bug reports, engaging in discussion of experience with currently implemented system features, reviewing and revising system documentation or community-centered artifacts, contributing intra- or inter-application scripts for gluing system capabilities together, and more. None of these ways of contributing are core developer tasks, nor are they traditional end-user tasks.

FOSS development projects enable large-scale, domain-specific learning

Openness, freedom of choice, and freedom of expression enhance opportunities for learning, education and science [Peters 2009]. The most commonly cited reason for joining a FOSS project is to *learn* — learn new skills, learn new problem or software development domains, learn from domain experts, learn from an apprenticeship, learn from participant observation, etc. [Scacchi 2007]. Also, large decentralized FOSS development programs like Google's *Summer of Code* (and also South Korea's *Winter of Code*) demonstrate new regimes for annually enabling hands-on participatory learning by thousands of students worldwide, independent of geographical location, national origin, or prior education, that facilitate informal software engineering and computer science education.

FOSSD is being used to help *teach* software development by doing, sharing, and collaborating, but for the most part, academic Computer Science programs have yet to adopt such learning opportunities. FOSS project contributors are also learning how to become a more effective project contributor, how to migrate into more development-oriented roles, and how to apply FOSS systems to new application areas including humanitarian assistance and relief. Similarly, global software development and global use of FOSS systems requires localization and cultural sensitivity, and it appears that FOSS systems can transfer between cultures more easily or more efficiently than proprietary software. Is this inevitable? Do FOSS systems have competitive advantages over proprietary software when moving into global markets, and can proprietary software product vendors overcome such advantages through further technological innovation? Last, can the development and deployment of FOSS systems for humanitarian applications provide a new strategy for engaging undergraduate students to take courses in CISE academic subjects [Morelli, Tucker, *et al.* 2009]?

FOSS systems are an engine of innovation

Successful FOSS systems and communities can grow at sustained exponential rates through ongoing contributions that realize continuous improvement and evolutionary adaptation [Deshpande and Riehle 2008; Koch 2005; Scacchi 2006]. FOSS has become an engine of innovation in many FOSS system user communities where users can become innovators [von Hippel 2001, Baldwin and von Hippel 2009], and where it is seen as a basis for enabling new opportunities to enter global software markets and challenge incumbent firms [Reding 2007]. The development of FOSS systems is a global socio-technical movement leading the way towards open science, open content, and open culture. But it is one of the few such movements, or perhaps the only one at present, that has CISE disciplines at its core.

We believe that FOSS systems are a game-changing engine of innovation of historic proportions that are transforming how people work together to develop complex systems (and systems of systems). Increasingly, the grand challenges of engineering research identified by the National Academy of Engineering (cf. http://www.engineeringchallenges.org) rely on the development of FOSS systems, such as the International Thermonuclear Energy Research (ITER) project for fusion research. Within some of these challenges, the development and experimentation with FOSS systems are likely to be central to research activities (e.g., advanced health informatics, secure cyberspace, enhanced virtual reality, and advanced personalized learning systems).

As we mentioned earlier, the Debian Gnu/Linux FOSS distribution may be the largest software system ever created, constituting more than 500M source lines of code. The

development and diversification of the core infrastructure to the World Wide Web and Internet primarily rests on FOSS systems and concepts (e.g., TCP/IP stack, network application protocols such as HTTP, FTP, SMTP, etc., Web browsers, and Web servers). No corporation or government enterprise currently appears capable of building and sustaining software systems of ULS size and complexity that can surpass what is being achieved with FOSS systems. But we lack the fundamental scientific knowledge to explain how and why this is so.

FOSS systems are transforming scientific research practice across disciplines

FOSS development processes, work practices, and project community dynamics are being put to work in R&D projects in the physical and biological sciences and various fields of engineering, and have also become the subject of research in the economic, legal, and social sciences [e.g., Balka, Raasch, *et al*. 2009; Bertelli, Bovo, *et al.* 2007; Lakhani, Jeppesen, *et al*. 2007; Raasch, Herstatt, *et al*. 2009; von Hippel 2001; Wilinski 2005]. Research is now underway in such diverse areas as Public Policy, Anthropology, Organization Science, Business/Management, Geography, Biology, Physics and E-Science, Imaging Science, Information Systems and more.

Open source systems are also being investigated in engineering and technology development efforts addressing hardware/digital product design [Rowe 2009 “open source business models”; Open Hardware Products 2010; Open Design 2010; Open Design Club 2010; Open Design of Circuits 2010]. However, some of these efforts have suffered from nominal or weak understanding of FOSS systems and technologies, while some early Computer Science-based studies of FOSS slight or ignore the social and community aspects that are essential to sustained FOSS projects.

Overall, it is clear that FOSS is a domain of CISE that is gaining the research attention of scholars in many scientific and cultural disciplines, as well as shaping their research agendas along with their industries.

FOSS development is helping to resolve outstanding problems in large-scale software engineering practice

FOSSD and SE are not in competition with one another. Nor is it fair to call one a variant of the other. FOSSD differs from SE in many ways, and it is these differences that are of greatest interest. For example, the widespread availability of FOSSD tools

has completely transformed the market for such tools, having driven down tool costs. The availability of automated tools for SE is now much less hampered by cost or lack of availability but the challenges of when, where, why, and who will adopt tool-based approaches to large-scale SE remain. SE also encourages the development of generic or reusable software components through careful design practices and testing refinements, while FOSS application development increasingly relies on pre-built FOSS system stacks (or "system of systems"). These are available across multiple platforms (cf. Bitnami.org; Asterisk.org), and encourage the emergence of collaborative communities that tailor, package, bundle, and install such systems as the basis for their competitive offerings [cf. Snow, Fjeldstad, *et al*. 2010].

Much industrial practice in large-scale software development is influenced by the Software Engineering Institute's Capability Maturity Model Integration (CMMI) framework (http://www.sei.cmu.edu/cmmi/start) as an approach to systematically improving the software development processes. FOSSD, in contrast, may or may not conform to the CMMI. However, FOSSD offers many means of continuously improving FOSS systems, and of developing and sustaining them within a decentralized community of practice. The CMMI framework was conceived to encourage the adoption of modern SE processes in various enterprises and government agencies. There is no widely accepted, comparable scheme for evaluating the maturity of a software development organization's capability with FOSSD processes. However, there is nothing that prevents the application and assessment of FOSSD processes in a commercial environment using the CMMI. Any perceived conflicts with FOSSD processes that exist arise from CMMI's assumed model of centralized software process and project management, control and planning, as opposed to FOSSD's assumed model of decentralized process management and self-organizing project control. If CMMI can be adapted to assess software development projects that operate in a decentralized, self-organizing and self-managed manner, then FOSSD projects could be assessed by CMMI, and it would be possible to certify FOSSD projects at different CMMI levels, as is done with SE project organizations. But until such time, the models and metrics for assessing SE practices appear ill-suited for FOSS systems and FOSS ecosystems, and thus it is unclear what kinds of models or measures are needed to help understand which FOSS systems are likely to succeed or fail, and which are of high-quality or low-quality in whose eyes or by what criteria.

FOSS systems are transforming the global software and IT industries

Every major IT and software company worldwide has been or is now investing in FOSSD projects in-house or off-shore [Agerfalk and Fitzgerald 2008]. But what this means is unclear, and likely varies by company. Some may see FOSS systems as a strategy for cost reduction, increasing pressure on competitors, or free-riding on the

efforts of others outside the company. For example, Dinkelacker, Garg, *et al.* [2002] described activities at Hewlett Packard that aimed to adapt the benefits of FOSS systems for internal use through the progressive introduction of FOSSD practices. They began with “inner source” (or “corporate open source” [Gurbani, Garvert, *et al.* 2010], the opening of all software source code behind the corporate firewall, then “controlled source” which restricts access to contracted partners, and finally to “open source,” where the community outside of HP was invited to participate and contribute. Elsewhere, informal sources indicate that major science research government ministries in the U.S., Europe, Japan, and others supporting software development are funding FOSSD projects that focus on evolving and expanding
enterprise IT to open cloud computing and open enterprise computing [2020FLOSS, 2010].

Growing numbers of national and regional governments, military/defense agencies, and ministries of education worldwide are establishing policies that encourage the development and deployment of FOSS systems [Lewis, 2010]. Sometimes these policies, especially outside of the U.S., are intended to mitigate the perceived dominance of proprietary software product companies in international markets. This may be to the disadvantage of U.S. software/IT companies, but help to stimulate the workforce development and learning of FOSS system practices in other nations.

Many of the largest Web generation businesses like Amazon, Yahoo, and Google, large financial services firms like Bank of America, Goldman Sachs, JP Morgan, Morgan Stanley, and Barclays Global Investors [Schmerken 2009], and national health care providers like Kaiser Permanente now develop and deploy online Web systems, middleware, or back-end servers that rely on FOSS systems.

The U.S. Department of Defense initially may have been hesitant to adopt FOSS systems that might contain source code contributed by foreign nationals outside the U.S., but now DoD policy and guidelines actively encourage and embrace selected FOSS systems as viable "off-the-shelf" choices [Herz, Lucas, *et al*. 2006, Hissam, Weinstock, *et al.* 2010, Scacchi and Alspaugh 2008]. Will the potential widespread adoption of FOSS systems within the DoD stimulate or erode the base of traditional defense contractors whose contracts increasingly depend on developing, deploying, and sustaining proprietary, closed source software-intensive systems?

FOSSD offers no remedy for the problems of large-scale SE. While a growing minority of SE projects succeed in producing viable and useful systems, the vast majority of FOSSD projects do not. Yet the comparatively small percentage of FOSS systems that do succeed seem to become very widespread, suggesting an ecological diversity approach to software system development in which only the strong survive and thrive. How to determine which will be strong is unclear, as is determining which socio-

technical conditions, project forms, communities of practice, or other software ecosystem parameters help increase the likelihood that a given FOSS system will thrive and become widespread.

FOSS systems are transforming governments, society, and culture

A small but growing number of scientific, cultural, and arts disciplines, as well as new government organizations in emerging arenas for collective action, are embracing the move towards more openness. One can now find a growing number of references to "open science," pointing to new work and institutional practices where openness, transparency, and peer production within decentralized organizational forms — all hallmarks of the open source system development paradigm — are the norm [Bradley 2007; Everts 2006; Kelty 2001; Swedlow and Eliceiri 2009]. The Public Library of Science (http://www.plos.org) has emerged as a leading source for publication of scientific research results that follow the practice of open science [cf. David 2004], where contributing researchers routinely provide not only journal articles, but open access to open data sets, and to FOSS tools used to analyze their data . Another example is the recent effort to overcome the "tragedy of the anti-commons" (Heller, 1998) in biomedical research through arrangements that encourage openness and open data sharing (e.g., [Kolata, 2010]).

The U.S. Department of Defense has begun to refocus its research in the development of command and control systems, as well as enterprise information systems, towards those that operate within decentralized edge organizations with systems based on open architectures and FOSS components [Alberts and Hayes 2003, Starrett 2007, Weathersby 2007, Scacchi and Alspaugh 2008, Hissam, Weinstock, *et al*. 2010]. This follows DoD's leadership as the first major government agency to recognize that mature FOSS technologies were being widely used within the DoD community for at least a decade, and that it would be a disadvantage to the mission of DoD to be restricted from using or building new systems with FOSS systems or components [MITRE 2002, Wheeler 2007]. Further, DoD policy now stipulates that mature FOSS technologies are to be treated as commercial off-the-shelf products that need not be redeveloped by its contractors when bidding on new system acquisition contracts [Wennergren 2009]

National, state, and municipal governments are also electing to adopt and deploy FOSS-based systems, as well as adopting open standards and open data formats, as a way to realize a more open government. FOSS also represents a low-cost approach to acquire and deploy common software system applications (e.g., personal productivity tools and office packages, Web browsers), software development and scripting tools (programming language environments, script interpreters, language development tools), back-end servers (Web servers, data base management systems, application servers,

networked file servers), operating systems (Linux, FreeBSD, OpenBSD) and middleware (network protocol handlers, data extraction, transformation, and loading tools).

People who develop and use FOSS systems share their knowledge and experience with others across FOSS projects in ways that collectively advance many such systems. However, “Open Source” is often used as a vague metaphor, and this gives rise to incorrect expectations and undermining the potentially positive influence of FOSS systems. While significant progress has been made in describing, analyzing and understanding FOSS systems, we need to extend and improve this research. Even with a solid understandings of how FOSSD works, we must anticipate how these techniques might apply in other domains, and the difficulties and challenges encountered when groups attempt to transfer aspects from open source to other domains.

Lastly, in addition to the most widely cited “open content” example, Wikipedia, there are efforts to take the ideas of openness and the innovation of open content licensing (CreativeCommons.org) and apply them in domains such as educational sharing and Collaboration, e.g., MIT Open Courseware and the broader Open Courseware Consortium (http://ocw.mit.edu; http://www.ocwconsortium.org/), and Rice University’s Connexions project (http://cnx.org/). These innovations are also being extended into cultural media creation as well [Cheliotis 2009, Hughes, Lang, *et al*. 2007; Lang, Shang, et al. 2007; OpenSourceCinema.org 2010].

Many key FOSS system projects are U.S. led

FOSS projects enable people from around the world to participate in software development projects to address their own interests and to facilitate technical skill development. The majority of project contributors are international (70% of FOSS developers are based in EU countries [Reding 2007]). However, many key FOSS projects like the Linux Kernel, Apache Web server, Mozilla/Firefox Web browsers, World Wide Web, OpenOffice/LibreOffice productivity suite, and Eclipse interactive development environment are led by core developers working in the U.S. Similarly, FOSS systems are the technological basis for a growing number of software-centered entrepreneurial start-up ventures in the U.S. [Augustin 2010]. So much FOSS system innovation originates in the U.S., as does much of the global software products industry, and many of the leading IT consultancies. But this lead is neither inevitable nor assured.

International competition is at hand. It is receiving increasing shares of national or regional government investment, upping the incentives to adopt globally created FOSS systems as alternatives to U.S. made software products.

Should U.S. national policy seek to stimulate research and development of FOSS systems? Should U.S. research policy encourage comparative study of FOSS system development practices and outcomes in the European Community, BRIC (Brazil, Russia, India, and China) countries, or other nations that facilitate FOSS system development efforts in non-English formats? Are FOSS development processes and practices globally common, or are there significant cultural and national differences? Does this serve to open or close international markets to future U.S. software products and services?

What have we learned so far about FOSS Systems? Observations on FOSS systems studies

At present, a small but growing community of FOSS system researchers in CISE and related disciplines are now engaged in a variety of empirical studies of FOSS system development processes, work practices, and project community dynamics to help understand what works, when, where, why and how in FOSS projects of different kinds. The results, challenges, and infrastructures emerging from these studies are highlighted throughout the remainder of this report. Our goal is to develop a new vision and research agenda for the FOSS system research community. Community members have individually addressed a number of interesting issues about the creation and use of FOSS systems. But research to date has not articulated an overall vision, nor does it systematically connect to national priorities like long-term stimulation of job growth, science-technology workforce development, the advancement of scientific knowledge, and overall economic growth. These need to be addressed in future studies.

The FOSS systems research community is growing across and within multiple disciplines including Computer Science, Software Engineering, Information Systems, Information Studies/Informatics, Human-Centered Computing, and others, as well as connecting to researchers in industrial research labs or large non-academic FOSS projects. The community is of a manageable size, making it feasible to bring together leading researchers and to disseminate a vision across research groups.

Empirical studies of FOSS system development are expanding the scope of what we can learn about how large software systems have or can be developed. In addition to traditional methods used to investigate FOSS systems like reflective practice, industry polls, survey research [Hertel, Neidner, *et al*. 2003], and ethnographic field studies, comparatively new techniques for mining software repositories ([Howison, Conklin, *et al*. 2006, Garg, Gschwind, et al. 2004, Gasser, Ripoche, *et al*. 2004, Robles, Gonzalez-Barahona, *et al*. 2004] and multi-modal modeling and analysis of the socio-technical processes and networks found in sustained FOSSD projects ([Scacchi, Jensen, *et al*.

2006, Scacchi 2007, Schweik, *et al.* 2010]) show that the empirical study of FOSSD is growing and expanding. Further studies will advance empirical computer science in fields like Software Engineering, which was previously limited by the restricted access to data characteristic of large, proprietary software development projects. Additionally, such studies help inform FOSSD projects in other scientific and cultural disciplines, and thus highlight the contribution of computer science research and education to those disciplines. Subsequently, empirical studies of software products, processes, projects and organizations will increasingly rely on data collected from FOSS development projects. Thus, these studies will increasingly be studies of FOSS system efforts.

The diversity and population of FOSS projects and multi-project repositories is unknown. There is great interest in the research community in a baseline and ongoing census of FOSS multi-project repositories. As FOSS projects collect, organize, and share the raw data of software development as an activity in their self-interest, then it behooves us within the research community to offer some guidance or incentives for these independent FOSS projects to contribute to such a census. Similarly, we need to articulate what benefits (e.g., socio-economic impact or intellectual contribution) the research community might offer in return to the FOSS projects that contribute to such a census.

- Data varies in *content*, with types such as communications (threaded discussions,chats, digests, Web pages, Wikis/Blogs), documentation (user and developer documentation, HOWTO tutorials, FAQs), development data (source code, bug reports, design documents, attributed file directory structures, CVS check-in logs) [Scacchi 2002, Scacchi 2007], and programming languages [Delorey, Knutson, *et al*. 2007].

- Data originates from different *types of Web-accessible online repository* sources [Deshpande and Riehle 2008, Hahsler and Koch 2005, Howison, Conklin, *et al*. 2006, Gao, Van Antwerp, *et al*. 2007, Mockus, Fielding, *et al*. 2002]. These include shared file systems, communication systems, version control systems, issue tracking systems, content management systems, multi-project FOSS portals (SourceForge.net, Freshmeat.net, Savannah.org, Advogato.org, Tigris.org, etc.), collaborative development or project management environments, FOSS code indexes or link servers (free-soft.org, LinuxLinks.com), search engines (Google.com/code, krugle.org, ohloh.net), and others. Each type and instance of such a data repository may differ in the storage data model (relational, object-oriented, hierarchical, network), application data model (data definition schemata), data formats, data type semantics, and conflicts in data model namespaces (due to synonyms and homonyms), and modeled or derived data dependencies. Consequently, data from FOSS repositories is typically heterogeneous and difficult to integrate beyond semantic hypertext linking [Noll

and Scacchi 1991], rather than homogeneous and comparatively easy to integrate.

- Data can be found from various spatial and temporal *locations*, such as community Web sites, software repositories and indexes, and individual FOSS project Web sites. Data may also be located within secondary sources appearing in research papers or paper collections (e.g., MIT FOSS research paper repository at http://opensource.mit.edu/), where researchers have published some form of their data set within a publication [Mockus, Fielding, *et al*. 2002, Scacchi, Jensen, *et al.* 2006, Wasserman and Capra 2007].

- Different *types of data extraction tools and interfaces* (query languages, application program interfaces, Open Data Base Connectors, command shells, embedded scripting languages, or object request brokers) are needed to select, extract, categorize, and prepare datafor further analysis [Garg, Gschwind, *et al*. 2004, German and Mockus 2003, Jensen and Scacchi 2006, Kawaguchi, Garg, *et al*. 2003, Ripoche and Gasser 2003, Robles, Gonzalez-Barahona, *et al*. 2004], as well as provide new kinds of tools and techniques for visualizing evolving software systems and the social networks that develop them [De Souza, Quirk, *et al*. 2007, Ogawa, Ma, *et al*. 2007, Ogawa and Ma 2008].

- Most FOSS project data is available as *artifacts or by-products* of development, usage, or maintenance activities in FOSS communities. These artifacts and byproducts are a critical part of the FOSS innovation process [West and Gallagher 2006]. However, very little data is directly available in forms specifically intended for research use. This has several implications for the needs expressed above [Gasser, Ripoche, *et al*. 2004, Robles, Gonzalez-Barahona, *et al*. 2006, Scacchi 2002, Scacchi 2007].

The open and public accessibility of data from FOSS project repositories and multi-project repositories like SourceForge.net, FLOSSmole, Google Code and others [cf. Howison 2009, Gao, Van Antwerp, *et al*. 2007, Garg, Gschwind, *et al*. 2004, Gasser, Ripoche, *et al*. 2004, Robles, Gonzalez-Barahona, *et al*. 2004] is providing a new, empirically grounded view of software technology and software development practice — a view that enables comparative, cross-sectional, and ecosystem level studies. These repositories and associated data collection, cleaning, analysis workflows, and dissemination tools point to the need for a more robust and widely available research infrastructure to support FOSS systems studies. This in turn means news kinds of research questions can be posed, and new knowledge can be discovered or created. This is needed if we are to overcome the gaps in scientific knowledge that we identify above and throughout the remainder of this report.

For example, repository-based studies of ongoing FOSS projects indicate that their software code base, functionality, development artifacts, developer contributions, and user base can undergo sustained exponential growth, apparently in contradiction to long-standing “laws of software evolution” which traditionally predict sub-linear, inverse square growth rates [cf. Capiluppi, Morisio, *et al*. 2004, Deshpande and Riehle 2008, Koch 2005, Scacchi 2006]. As such, the kind of research questions that follow ask what model or theory accounts for the super-linear evolution of FOSS systems. Another example: are there software patterns that constitute a kind of “software genome” that characterize the evolutionary mechanisms of different families of independently developed FOSS systems? Similarly, are the critical software components or functions that lie at the heart of different software families FOSS, and does such software represent a critical evolutionary or security vulnerability (e.g., the glibc library is commonly bound with FOSS coded in the C programming language)? Also, what development processes best characterize FOSS projects that demonstrate sustained exponential growth of their code and functionality base, as well as the growth of the number of contributors, but with comparable growth/decline of software quality? Last, what can we learn about the evolution of FOSS systems across multiple releases, across multiple platforms, and across different independently developed variants? Exploring questions like these requires data drawn from multiple FOSS projects or project repositories, and this data is now available. As such, we are on the verge of possible discontinuous advances in our knowledge about software, based on empirical studies of FOSS.

Articulating new knowledge of software products, processes, practices, and organizational forms depends on careful and systematic empirical study of FOSS project data. However, this data is not easy to collect or analyze. As such, there is a need to articulate practices for the curation of FOSS project data in forms that increase the likelihood of data use and analysis by people in different disciplines and settings. There is also a need to capture data provenance as well as data annotation and data analysis workflow tools and techniques. Other science disciplines have recognized similar needs. Such research can be conducted as both discipline-specific and cross-discipline. At present, the FOSS research community has little practice with and access to these tools and techniques, and has little incentive to take on their application or reinvention.

The commercial software products and service industry in the U.S. is in an awkward strategic position regarding whether or how to take advantage of FOSS systems, or the results arising from studies of FOSS development data. Software product companies like Microsoft seem hesitant about what to do about FOSS, while software service companies like Google seem to embrace FOSS (as do computer vendors like IBM and Oracle-SUN). But all software companies could benefit competitively from empirical studies of FOSS products, processes, practices, and organizational forms .

Lastly, companies like Google, SUN/Oracle, Hewlett-Packard, IBM, and Microsoft have established a community of FOSS development projects under their corporate sponsorship. These projects are sponsored to increase the pool of future software developers who might become skilled in the use of other products or services offered by these vendors. However, these companies may also benefit by helping to cultivate and recruit these developers into their commercial software workforce. FOSS development projects develop a workforce skilled in complex software systems development. These projects provide a situated, real-world experiment in informal software engineering education that often takes place outside of traditional higher education. However, "data" from these educational experiences is generally not open, nor publicly available, as it is said to be sensitive, confidential, and proprietary. Thus it is unclear how well these informal experiments work, and how they can be improved from a corporate as well as from an academic perspective. Perhaps the academic software research and engineering community can be brought together with the commercial software industry through a government sponsored forum to articulate how best to advance U.S. socio-economic and scholarly interests in the software community.

Where is the action? Areas and impacts for FOSS systems research

Based on discussions and debate at FOSS research workshops in 2008-2010, four areas of research seem to offer the most promising and challenging problems to investigate and resolve in the next 5-10 years. These areas for FOSS systems research are:

- *Processes, practices, and project forms* — what are the development processes, work practices, and alternative project organizational forms that give rise to successful FOSS systems? What works where, when, why and how, and for whom?

- *Collaboration* — how does the practice of developing large or ULS software systems depend on the collaborative work practices and communities of practice found in successful FOSS system projects?

- *Ecosystems* — how do FOSS systems emerge within a complex, decentralized web of people, artifacts, practices, and other infrastructural resources where most FOSS projects fail to take root and thrive? How do those few that do succeed become widespread and transform industry, government, or science practices?

- *Evolution* — how can successful FOSS systems continue to grow and develop across ever larger communities of developer-users at sustained exponential rates? To what end, and following what processes?

The next section of this report examines each of these four areas to identify what is known so far, what are some of the outstanding research questions that need to be addressed, and what research resources and infrastructure are needed to help us overcome our gap in scientific knowledge of FOSS systems. The third section specifies FOSS system research community needs for resources and information infrastructures (or cyberinfrastructure) to explore and articulate scientific knowledge that is missing but within reach for further research into FOSS systems.

Finally, the fourth section identifies four areas in which FOSS systems research may have an impact. The impact in these areas helps to substantiate where investment in FOSS system research is likely to be most transformative, and demonstrates why an investment in FOSS systems research and research infrastructures are a critical national need. These areas are:

S*oftware development* — the development of reliable large, very-large, or ULS software-intensive systems requires more than robust, formalized, and mathematically grounded approaches to SE. The engagement of decentralized, global communities of practitioners who participate in and contribute to the development, use, and evolution of software system tools, online artifacts, and other information infrastructure resources is also necessary. The development of software-intensive systems at large-scale and beyond needs to be recognized as something now essential to the advancement of science, technology, industry, government, and society across geographic borders and cultural boundaries.

E*ducation and learning* — we need to educate students and the public to understand how best to create, access, study, modify, and share complex systems that are open and liberating. Widespread information resources, development processes, work practices, and online content that are free and open, rather than restricted to those who can afford to access them, will provide a new baseline for transforming learning in the sciences, industry, and democratic government.

I*nnovation* — engines of innovation for advancing science, technology, and engineering in industry, government, and society at large are few and far between. FOSS systems development is emerging as an engine whose openness encourages invention and reinvention, knowledge sharing and crowd-sourcing. It offers lower cost access to high capability information technologies that are transparent and open for widespread public access, study, modification, experimentation, ad hoc or systematic integration, repackaging, and redistribution.

FOSS systems can stimulate societal advances, innovations, and progressive transformations when their public access is assured and protected.

S*cience, industry, and government* — many national challenges for science and engineering depend on the development of a new generation of complex, software-intensive systems. Advances in enterprise information systems that realize new ways to streamline operations, create products and services and more stimulating jobs and workforce development opportunities, depend on faster, better, and cheaper software systems. Helping to make regional and national government agencies more transparent, open, and trustworthy requires public access to information systems that are easy to access, open for study and open to citizen participation. FOSS systems are the most likely technology that can realize these societal needs.

Subsequently, the remainder of this report examines the areas and impact of future research into free/open source software systems, as well as the infrastructures needed to conduct such research.

Part II

The Current State of FOSS

FOSSD Processes, Practices, and Project Forms

Overview

At least since Conway stated his famous observation that software artifacts and software project organizations appear to have interrelated structures [Conway 1968], we have recognized that processes, practices, and project organization forms are critical elements of software development efforts. Over the years, researchers have made many attempts to define, measure, and formalize software processes, with the aim of making them more rational, predictable, explainable, controllable, repeatable and transferable. Below we detail how and why understanding the processes, practices and forms of FOSS system development (FOSSD) contributes to the science of *complex systems*, what the critical perspectives and unknowns are, and what systematic research efforts are needed to fill in the essential missing knowledge.

Our scientific research goals

We are addressing how to best understand complex systems; FOSS systems and FOSSD projects constitute complex systems. A central part of this science is the ability to:

- ***Explain*** how and why FOSSD practices and processes behave, work and fail.
- ***Rationalize*** FOSSD practices with clear metrics of utility and generalized models with pragmatic value.
- ***Predict*** progress, development, and breakdown of FOSSD practices and processes.
- ***Control*** the trajectories and outcomes of FOSSD processes and practice.
- ***Evolve*** FOSSD practices and processes to incorporate new methods, standards, tools, and participants, to adapt FOSS practices to changing constraints and opportunities, and to improve them over time.
- ***Transfer*** and reuse FOSSD practices, processes, and project forms in new locations and projects, with different software and participants.

We envision FOSSD practices that are fully explainable, rational, predictable, controllable, evolvable, and transferable so that they can be employed effectively, continuously improved and adapted to new contexts, given situation-specific opportunities and constraints. We seek the scientific knowledge now missing to understand how, where, and when to employ FOSSD projects to build software in different application areas of science, industry, and government, as well as for the information infrastructures of the future. We need to understand in what contexts given FOSSD practices work best and worst, and what contextual "package" or ecosystems they entail.

The traditional view of software development processes, practices, and projects

Much is known and continues to be learned about how best to engineer complex software systems. Software Engineering (SE) is an academic discipline and industrial practice that seeks to rationalize and control the development of complex software system products and services. SE is a process by which an individual or team organizes and manages the creation of a software-intensive system, from concept through one or more releases. The *Software Engineering Body of Knowledge* [SWEBOK 2004] identifies a number of core engineering activities whose science and mathematical bases serve as its foundational principles. SE offer basic principles for software development that include tools and techniques for abstraction, modularity, architectural coupling and dependency analysis, testing and integration, internationalization and localization. These principles are further explained for educational purposes in textbooks on the subject, such as Sommerville's [2006] *Software Engineering, 8th. Edition*. A review of these materials finds that central software development activities should center around the following activities:

- *Software requirements* — identifying the problems a new software system is supposed to solve, its operational capabilities, its desired performance characteristics, and the resource infrastructure needed to support system operation and maintenance.

- *Software design* — software architecture design defines the interconnection and computational resource interfaces between system subsystems, components, and modules in ways suitable for their detailed design and overall configuration management. Detailed software component or module design defines the procedural methods through which the data resources within the modules of a component are transformed from required inputs into provided outputs.

- *Software construction* — codifies the preceding software specifications into operational source code implementations and validates their basic operation.

- *Software testing* — affirms and sustains the overall integrity of the software system's architectural configuration through verifying the consistency and completeness of implemented modules, verifying the resource interfaces and interconnections against their specifications, and validating the performance of the system and subsystems against their requirements.

- *Software maintenance* — sustains the useful operation of a system in its host/target environment by providing requested functional enhancements, bug/fault repairs, performance improvements, and conversions or ports to new platforms.
- *Software configuration management* — manages the software architecture over time as components, modules, interconnections, and interfaces are subject to change and incremental refinement during software system construction and maintenance.

- *Software engineering management* — focuses on providing the managerial planning, scheduling, budgeting, staffing, and organizing of software developers who are employed to develop software systems for users by performing the preceding activities.

- *Software engineering process* — the overall framework for how to organize and manage a software development project so as to produce a usable system that satisfies user requirements by directing the performance of software developers using tools and related resources that take provided data sources and transform them into valuable information products (e.g., reports, information displays) or services through the system being developed.

- *Software engineering tools and methods* — fully or partially automated software utilities, along with documented techniques for their proper usage, whose purpose is to facilitate the engineering of complex software systems.

- *Software quality assurance* — tools and methods (inspections, reviews) that seek to assure that user requirements are satisfied by the system development effort across the system's life cycle.

Over the past fifty-plus years, these activities have been organized, packaged, and refined through various “software development methodologies” that have been identified as: the Waterfall model, Prototyping, Spiral Development, Iterative and Incremental Development, Rapid Application Development, Object-Oriented Development, Top-

down programming, Unified Process Model, Agile Software Development, Integrated Methodology Software Development, Extreme programming, Rational Unified Process (RUP), and others. Leading SE scholar Barry Boehm, at the University of Southern California, has recapitulated the historical progress of such development methodologies in his review of software engineering from the 1950's to 2010's, as summarized in the following figure [Boehm 2006]:

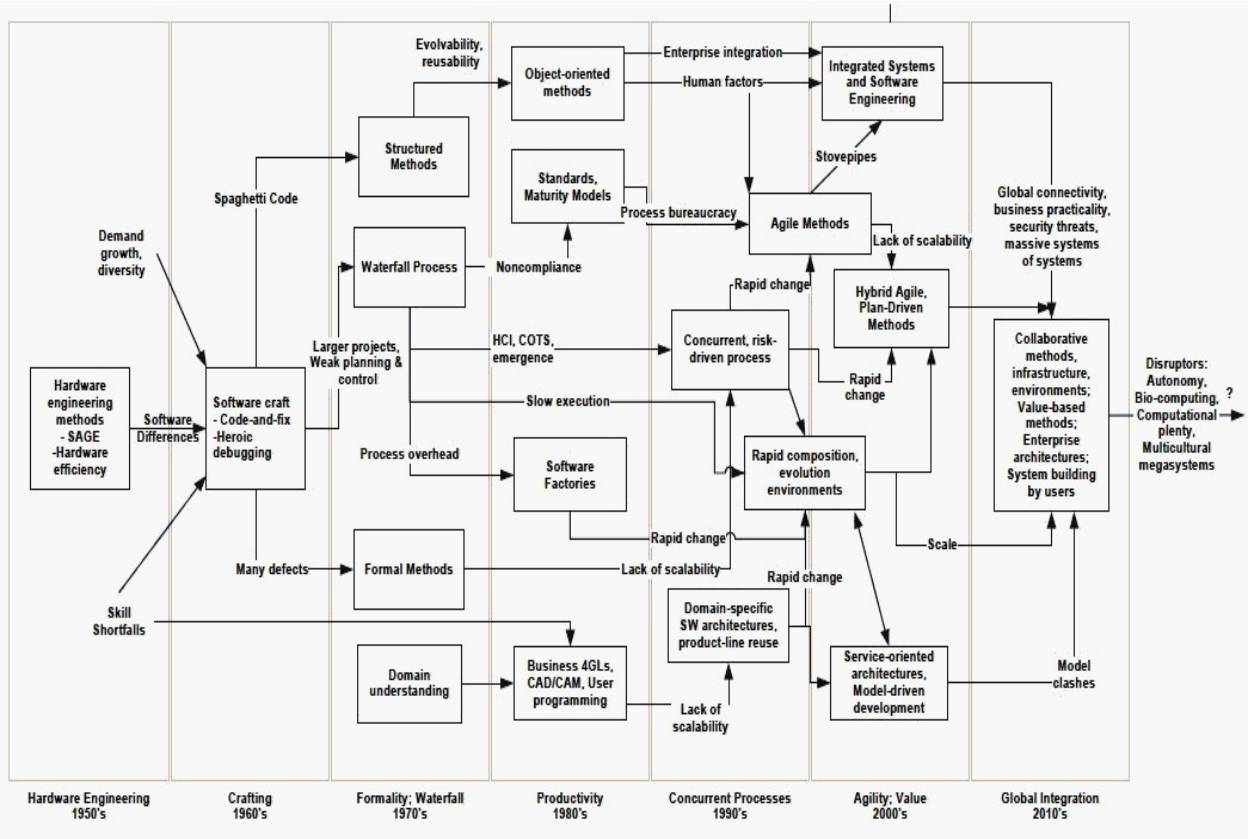


Figure 1: The evolving history of progress in development methodologies for engineering software systems [Boehm 2006].

FOSSD does not appear as a development methodology in this view. Boehm [2006] indicates that FOSSD would be a concurrent engineering process for developing software systems, though without much detail for how such processes are conceived or operate in practice. So is FOSSD just another development methodology, or is it something else? Let us review some of what has been learned through empirical studies of FOSSD processes, practices, and projects.

What are FOSS Development processes, practices, and projects and how do they differ from those traditional to Software Engineering?

A significant hallmark of FOSSD is that the *source code* is open and available for remote access to others with few constraints. Beliefs, values, and norms associated with freedom to choose the ways and means for developing FOSS systems, along with freedom to express, justify, and critique such choices in an accountable and traceable manner, are found in many successful and self-sustaining FOSSD projects. FOSS systems sometimes add or remove similar freedoms or copyright privileges depending on which FOSS copyright and end-user license agreement is associated with a particular FOSS code base [Fontana, Kuhn, *et al.* 2008; Rhoten, Powell, *et al*. 2007; Rosen 2004]. More simply, free software is always available as OSS, but OSS is not always free software. This is why it is often appropriate to refer to FOSS or FLOSS (L for *Libre*, where the alternative term “libre software” has popularity in Europe and elsewhere) in order to accommodate two similar and often indistinguishable approaches to software development. Subsequently, for the purposes of this article, our focus is on collaborative FOSSD processes, practices, and project dynamics.

The focus and primacy of FOSSD activities are different from those of SE. For example, few if any software requirements and designs for FOSS systems are formally specified or clearly documented in separate documents that are identified as such; instead, requirements are asserted in dispersed online artifacts after, rather than before, they are implemented [Noll 2008; Scacchi 2002; Scacchi 2009].

Boehm [2006] reported that the top three reasons software engineering projects fail are (a) lack of user input; (b) incomplete requirements and specifications; and (c) changes to those requirements and specifications. FOSS system requirements cannot be complete if they are not specified or asserted until after they are implemented, which means they can be fluid, dynamic and subject to change. Similarly, FOSSD projects rarely employ a software engineering project management scheme entailing schedules and budgets, and instead rely on informal, self-organizing agreements and communications.

The informality of project management creates effective project organization and software production. In one study, this was designated as “virtual project management,” that is, project management without the budget, schedule, and staffing, but with effective, lightweight governance and coordination mechanisms [Scacchi 2004; Jensen and Scacchi 2010]. Further, critical FOSSD processes seem to center around (a) motivating, recruiting, and migrating new FOSSD participants into different developer, contributor, or user roles; (b) forming social networks, multi-project alliances, and project communities; and (c) evolving FOSS systems within multi-project software ecosystems

[Scacchi, Feller, *et al*. 2006]. Furthermore, as demonstrated elsewhere [Scacchi, Jensen, *et al*. 2006], FOSSD processes are decentralized and spread across loosely coupled, geographically dispersed OSS developers. Nonetheless, FOSSD processes can be discovered using process mining tools and techniques [Jensen and Scacchi 2007, 2010], defined and modeled [Jensen and Scacchi 2005], computationally analyzed and enacted [cf. Noll and Scacchi 2001], and thus potentially optimized and redesigned [cf. Scacchi 2001].

FOSSD is mostly not about SE, at least not as SE is portrayed in modern SE textbooks [cf. Sommerville 2006]. FOSSD is not SE done poorly. It is instead a different approach to the development of software systems in which much of the development activity is openly observable and development artifacts are publicly available over the Web. Substantial FOSSD effort is directed at facilitating collaboration among developers (and sometimes end-users), generally without traditional software engineering project management. FOSSD is also oriented towards the joint development of an ongoing community of developers and users concomitant with the FOSS system of interest.

When the principles of modern SE are applied in industrial centers or in government system acquisition programs, a number of learned lessons are generally recognized as “best practices” for developing software system products or services through SE [SWEBOK 2004; Sommerville 2006]. In contrast, six areas can be examined to discover the practices of FOSSD.

1) Some FOSSD projects embrace modern SE principles, but may do so through practices different from those found in industry best practices. An example can be found in the hundreds of FOSSD projects associated with the Tigris.org FOSS-SE community. Exhibit 1 presents a view of the best practices the Tigris.org project community has identified.

Key Open Source "Best Practices" supported in the Tigris project

Technical Communication

Software developers spend a large part of their time communicating with each other. Clear and effective technical communications are needed to keep the team in synch and to allow individuals with key knowledge to apply that knowledge where it is needed.

One tenet of the open source community is that techincal communications should take place in public forums. Mailing lists are the backbone of open source communications. Beyond that, open source projects need support for precisely communicating technical details and for group decision-making.

Version Control, Document Management, and Distribution

Every software development project needs version control.

Open source projects need strong and flexible support for many concurrent developers working on overlapping sets of files. Open source projects also need standard templates for design documents, technical documentation, and end user documentation.

A well organized web site is required to distribute these documents. But, the nature of open source projects demands that these files are globally accessible, and that administrative overhead be minimized and responsibility be spread out over the members of the development community.

Quality Assurance

Open source software products have achieved a remarkable degree of quality. This is something that the closed source development world has found to be one of the most difficult and costly aspects of software development.

Open source software is high quality for three main reasons:

1. Developers are self-selected by their interest andknowledge of the application domain,
2. requirements are tacitly understood by developers who are themselves users of the software,
3. technical communications (including bug reports) are conducted in public. The public nature of open source helps developers take pride in their successes and think twice before releasing faulty code.

Furthermore, debugging is an activity that is more effective when split among many people who have diverse knowledge

Build and Test Management

Development activity on open source projects is constant and concurrent among many developers. Sometimes the changes made by one developer interfere with those made by another developer.

Often the version control practices used in the open source community resolve these conflicts, but sometimes they do not. Frequent automated builds and tests of the software being developed are powerful tools that help catch logical conflicts early

Project Management

Every project needs explicit goals, resources, and a schedule.

The open source community has addressed these issues in a uniquely flexible way. Shared "to do" lists keep track of tasks that need to be done. Personal "to do" lists keep developers on track. Milestone lists set flexible deadlines for individual features based on feedback from users and developers

Knowledge Management

Knowledge is the valuable resource that sets experienced developers apart from novice ones.

Sharing knowledge effectively has been key to the success of the open source community. Explicitly managing knowledge can help reduce the learning curve for novices which reduces the barriers to entry for potential contributors while automatically keeping the load on the experts down to a minimum in terms of training others.

Exhibit 1: Best practices advocated for Tigris.org open source software engineering projects (source: http://www.tigris.org/community/vision/best_practices.html, accessed June 2010).

But one FOSSD project's declaration of its best practices does not necessarily indicate whether these practices are best for other FOSSD projects. Consequently, research questions arise such as:

- What is the best way to determine which practices are best for a given FOSSD project before or during its development?

- How should studies of FOSSD practices be designed so that a small study sample will allow for generalizable results?

2) There are FOSSD projects that are supported by, or organized within, industrial or international software development centers. Examples here include the NetBeans and Eclipse FOSSD projects that are both developing Java-based interactive development environments (IDEs) based in part on the corporate support respectively from Oracle-SUN (*NetBeans*) and IBM (*Eclipse*). Similarly, Dresder Kleinwort supports the development of the *OpenAdaptor* middleware service which in turn is employed by both its competitors in the international banking industry, as well as non-competing business partners like Hewlett-Packard. Companies like Sun, IBM, HP, SAP, and Microsoft all now support dozens of FOSSD projects, including some of their key, high revenue product lines (e.g., HP Inkjet and Laserjet Printer now use FOSS printer drivers) and employ FOSS systems that were derived from their own proprietary, closed-source, in-house software. Finally, international cooperatives like the European Space Agency (ESA) have adopted standardized processes for developing open source software systems for mission-critical space applications like *TerraSAR* [Peccia 2007] that are built on the open source spacecraft operating system, SCOS 2000 [Kaufeler, Jones, *et al*. 2001], that conform to international software engineering standards established by the ESA [Aldea, Jones, *et al*. 2003; ESA 2007]. These conditions raise the questions:

- How does corporate sponsorship or large enterprise involvement facilitate, impede, or otherwise transform a FOSSD project?

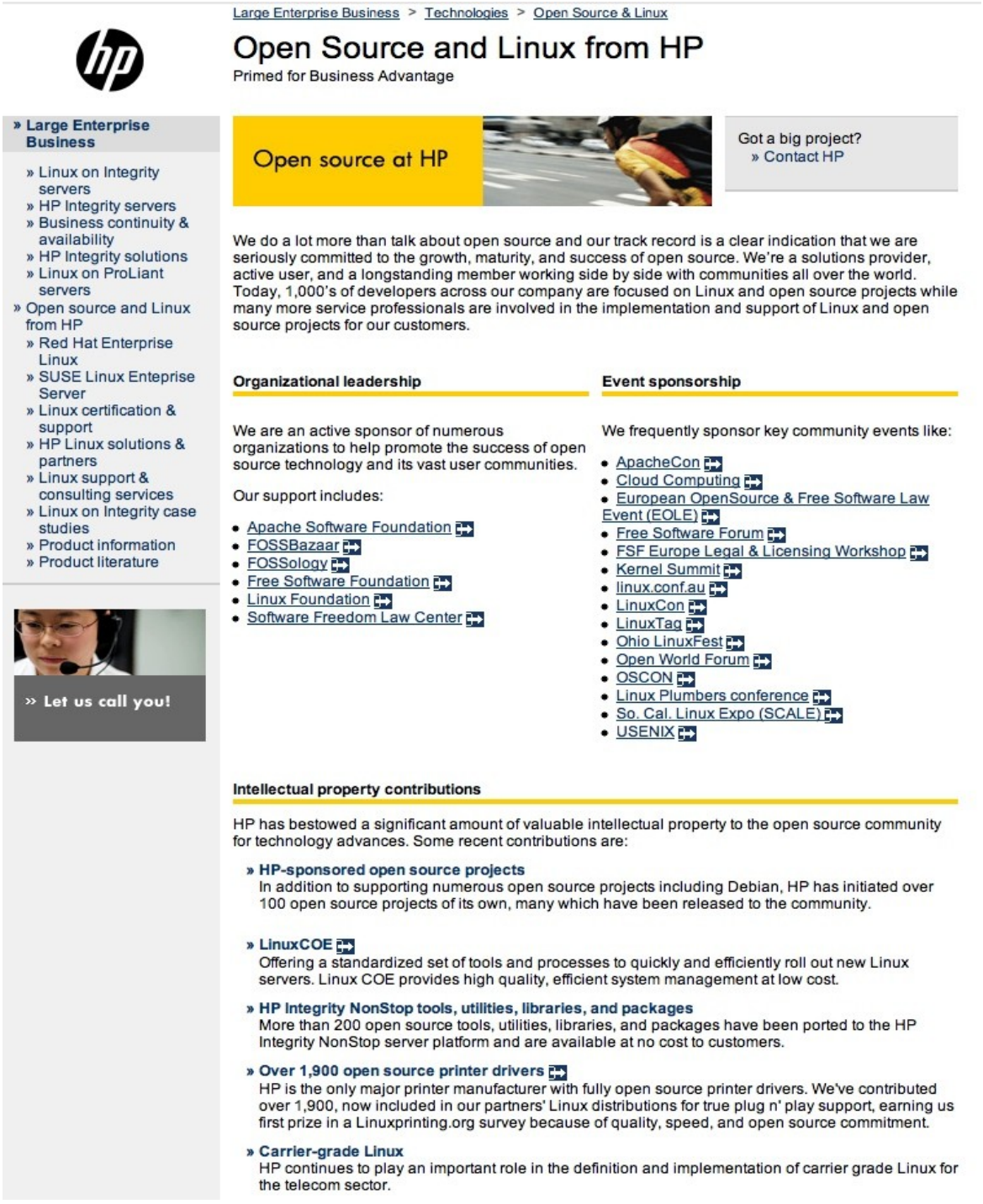
- What do large firms or enterprises learn from participating in large FOSSD projects?

3) FOSSD project management environments like SourceCast™ (SC) from Collab.Net, and Corporate Source (CS) from Zee Source are the products of commercially oriented FOSSD projects that have evolved into Web-based project management environments for collaborative software development [Augustin, Bressler, *et al*. 2002]. These environments are not IDEs like NetBeans or Eclipse, though they could be made to interoperate with them. These environments are non-free commercial products

marketed primarily to large corporations that may have dozens of organizationally dispersed software development projects underway at any time.

- Under what conditions will FOSSD tools, techniques, and interactive development environments overcome and dominate the market for software development tools?

Hewlett-Packard, for example, has now made its investment in FOSS system technologies a central part of its business strategy and marketing efforts, as indicated in Exhibit 2.



Large Enterprise Business > Technologies > Open Source & Linux

Open Source and Linux from HP

Primed for Business Advantage

» Large Enterprise Business
- » Linux on Integrity servers
- » HP Integrity servers
- » Business continuity & availability
- » HP Integrity solutions
- » Linux on ProLiant servers

» Open source and Linux from HP
- » Red Hat Enterprise Linux
- » SUSE Linux Enteprise Server
- » Linux certification & support
- » HP Linux solutions & partners
- » Linux support & consulting services
- » Linux on Integrity case studies
- » Product information
- » Product literature

» Let us call you!

Open source at HP

Got a big project?
» Contact HP

We do a lot more than talk about open source and our track record is a clear indication that we are seriously committed to the growth, maturity, and success of open source. We're a solutions provider, active user, and a longstanding member working side by side with communities all over the world. Today, 1,000's of developers across our company are focused on Linux and open source projects while many more service professionals are involved in the implementation and support of Linux and open source projects for our customers.

Organizational leadership

We are an active sponsor of numerous organizations to help promote the success of open source technology and its vast user communities.

Our support includes:

- Apache Software Foundation
- FOSSBazaar
- FOSSology
- Free Software Foundation
- Linux Foundation
- Software Freedom Law Center

Event sponsorship

We frequently sponsor key community events like:

- ApacheCon
- Cloud Computing
- European OpenSource & Free Software Law Event (EOLE)
- Free Software Forum
- FSF Europe Legal & Licensing Workshop
- Kernel Summit
- linux.conf.au
- LinuxCon
- LinuxTag
- Ohio LinuxFest
- Open World Forum
- OSCON
- Linux Plumbers conference
- So. Cal. Linux Expo (SCALE)
- USENIX

Intellectual property contributions

HP has bestowed a significant amount of valuable intellectual property to the open source community for technology advances. Some recent contributions are:

» **HP-sponsored open source projects**
In addition to supporting numerous open source projects including Debian, HP has initiated over 100 open source projects of its own, many which have been released to the community.

» **LinuxCOE**
Offering a standardized set of tools and processes to quickly and efficiently roll out new Linux servers. Linux COE provides high quality, efficient system management at low cost.

» **HP Integrity NonStop tools, utilities, libraries, and packages**
More than 200 open source tools, utilities, libraries, and packages have been ported to the HP Integrity NonStop server platform and are available at no cost to customers.

» **Over 1,900 open source printer drivers**
HP is the only major printer manufacturer with fully open source printer drivers. We've contributed over 1,900, now included in our partners' Linux distributions for true plug n' play support, earning us first prize in a Linuxprinting.org survey because of quality, speed, and open source commitment.

» **Carrier-grade Linux**
HP continues to play an important role in the definition and implementation of carrier grade Linux for the telecom sector.

Exhibit 2: Corporate support for FOSS at Hewlett-Packard Corporation, http://h71028.www7.hp.com/enterprise/cache/599999-0-0-0-121.html, accessed 29 June 2010.

Accordingly, information technology and services companies like Hewlett-Packard, IBM, Nokia, Novell, Oracle-Sun, and others have adopted FOSSD project management environments for use behind the corporate firewall [Dinkelacker, Garg, *et al.* 2002, Gurbani, Garvert, *et al.* 2010], or to support corporate sponsored FOSSD projects like *NetBeans* or *Eclipse*.

These FOSSD projects follow practices that arise from the use of the tools, services, and transactions for collaborative software development that these vendors offer. A view of the project management activities, services and capabilities supported by SC and CS appear in Exhibit 3. It should be noted that most FOSSD projects do not employ all of these capabilities.

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: Common set of software product development, technical communication, and project management tools/capabilities available for use within FOSSDs in a commercial environment [Augustin 2002].

Given the growing diversity of companies making public their reliance on FOSSD strategies in supporting development of their commercial products and services, questions such as these may follow:

- Is the adoption of FOSSD tools and techniques inevitable for commercial software/IT firms?Will those that fail to adopt them be at a significant commercial disadvantage in national or global markets?

- How can FOSSD tools and techniques be used within large enterprises to maintain or revitalize software system applications originally developed 10-30 years ago?

- How will the use of FOSSD tools and techniques in large enterprises evolve over time?

- What are the risks of relying on FOSSD tools and techniques for large Enterprises? How do such risks change over time?

4) Empirical studies collect and analyze software development practices and processes within FOSSD projects [e.g., Scacchi 2007]. These studies have produced quantitative results that characterize FOSS 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 FOSS development and evolution.

5) FOSS developers are typically end-users of the FOSS they develop [Scacchi 2002; Scacchi 2004; von Hippel, von Krogh, et al. 2003; von Hippel 2005; Ye, Nakajoki, et al. 2005], and other end-users often participate in and contribute to FOSSD efforts as non-core developers. There is widespread recognition that FOSSD projects produce high quality and sustainable software systems that can be used by thousands, even millions of end-users [Mockus, Fielding, *et al*. 2002]. It can't be assumed that FOSSD processes are the same as those used in modern SE projects [cf. Sommerville 2006]. While FOSSD approaches might be used within an SE project, nothing suggests that SE projects typically practice FOSSD methods. What is known about SE processes and practices may not be equally applicable to FOSSD practices without some explicit empirical justification. Thus, it is important to ask:

- What are the best ways to identify FOSSD processes in use in FOSSD projects?

- Under what conditions will FOSS developers prefer to follow a recommended process to guide their development efforts, in order to improve their productivity or software quality?

- What are the dominant development, tool acquisition, and project community management processes that are in use in FOSSD projects?

6) The Software Engineering Institute at Carnegie Mellon University is well known for its software development process improvement framework, the Capability Maturity Model Integration [CMMI 2006]. CMMI is a scheme for evaluating an SE organization's mature ability to develop software. CMMI encourages certain software engineering practices regarding development processes, tool use, and documentation of project activities. The

CMMI is also an internationally recognized scheme that seeks to improve software product quality through adoption of best practices in software engineering [SWEBOK2004]. The CMMI has been promoted and adopted throughout the U.S. Federal Government, as well as in hundreds of large corporations. What do we know about how FOSSD projects fit into or conflict with the CMMI?

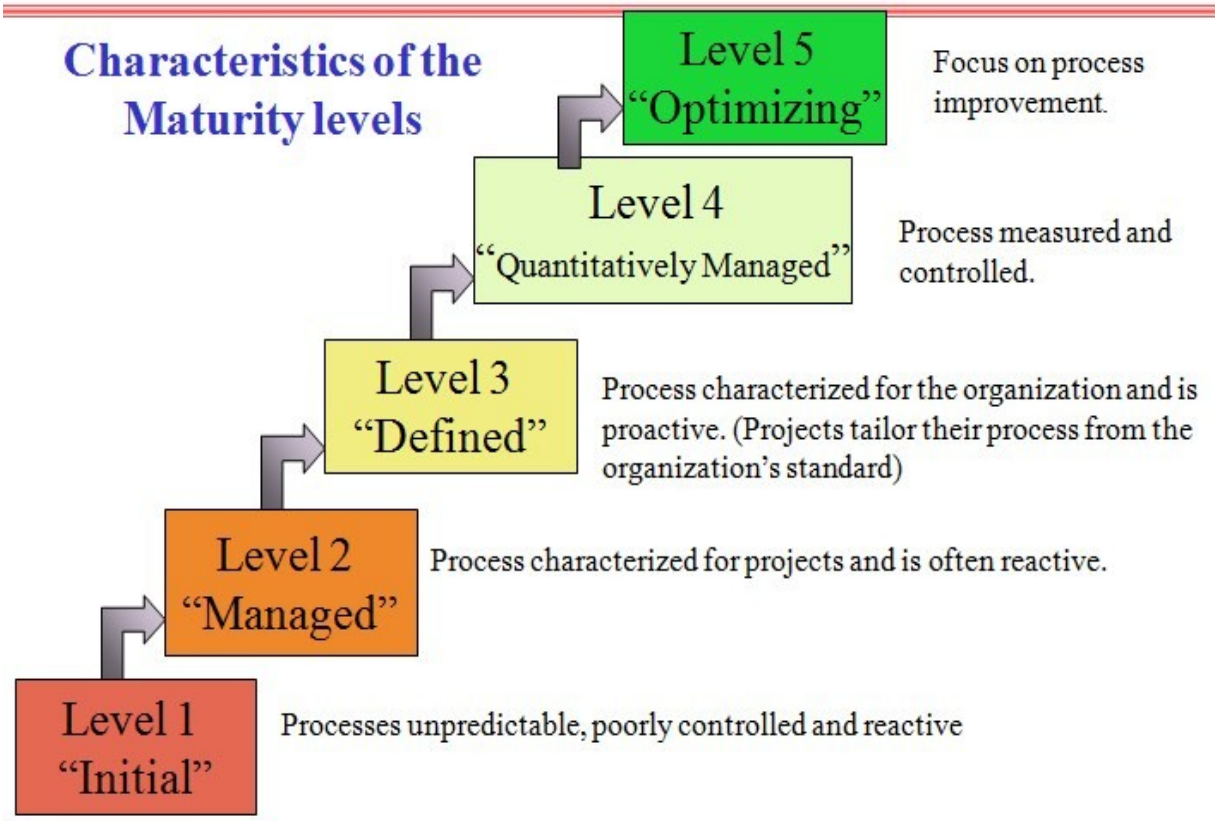


Exhibit 4: The Capability Maturity Model Integration (CMMI) levels.

Exhibit 4 provides a map that identifies and lays out the five CMMI levels, and process activities that characterize them, while Exhibit 5 outlines key process activities for each of the five levels.

Abbreviation	Name	Area	Maturity Level
REQM	Requirements Management	Engineering	2
PMC	Project Monitoring and Control	Project Management	2
PP	Project Planning	Project Management	2
SAM	Supplier Agreement Management	Project Management	2
CM	Configuration Management	Support	2
MA	Measurement and Analysis	Support	2
PPQA	Process and Product Quality Assurance	Support	2
PI	Product Integration	Engineering	3
RD	Requirements Development	Engineering	3
TS	Technical Solution	Engineering	3
VAL	Validation	Engineering	3
VER	Verification	Engineering	3
OPD	Organizational Process Definition	Process Management	3
OPF	Organizational Process Focus	Process Management	3
OT	Organizational Training	Process Management	3
IPM	Integrated Project Management	Project Management	3
RSKM	Risk Management	Project Management	3
DAR	Decision Analysis and Resolution	Support	3
OPP	Organizational Process Performance	Process Management	4
QPM	Quantitative Project Management	Project Management	4
OID	Organizational Innovation and Deployment	Process Management	5
CAR	Causal Analysis and Resolution	Support	5

Exhibit 5: Key Process Areas for the CMMI

For the most part, FOSSD and SE are mutually exclusive, alternative approaches to building complex software systems. The CMMI framework was conceived to encourage the adoption of modern SE processes primarily in large, centrally located and organized

corporate software development efforts. Evaluating FOSSD in terms best suited for SE will likely result in FOSSD being categorized as “ad hoc” or “chaotic” (designating a CMMI Level 1), even though FOSSD can sometimes produce better results than those achieved in a commercial environment with higher level CMMI software process characterizations. There is nothing fundamental that prevents the application and assessment of FOSSD based projects in a commercial or governmental environment using the CMMI.

Any perceived conflicts with FOSSD arise from CMMI’s assumed model of centralized software process and project management.

There is no widely accepted scheme comparable to CMMI for evaluating the maturity of a software development organization’s capability with FOSSD. If CMMI can be adapted to assess software development projects that operate in a decentralized manner, FOSSD projects could be assessed and certified at CMMI levels.

As FOSSD projects rarely if ever provide explicit software development processes or process models, it would seem as if they could not realize a CMMI Level 3 rating, There is no “defined” FOSSD process, as FOSSD processes vary across projects[Jensen and Scacchi 2005, 2007, 2010]. Early indications are that software development projects that try to combine traditional software project management practices with FOSSD eventually produce projects consistent with either traditional efforts or FOSSD efforts, but do not gain from their combination. In fact may lose quality through joint sub-optimization and technical conflicts [cf. Rosenberg 2008].Why this is so is yet another area in which scientific knowledge is lacking.

There are individual proposals for how to construct a FOSSD evaluation scheme, but they generally focus attention on organizational capability, and comparatively few organizations developing software employ FOSSD. Those that do have not sought to certify their capability. Instead, there is an emerging scheme that focuses on assessing the readiness of FOSS systems for development and use in business enterprises [Wasserman, Pal, *et al*. 2006]. Consequently, this leads to such deeper questions as:

- In what ways are software process maturity models relevant to improving FOSSD projects?

- Under what conditions are FOSSD processes and practices more effective at producing high quality software systems compared to organizations that rely on process maturity models?

What else do we know about FOSS processes, practices, and project forms?

FOSSD is not a panacea for developing complex software systems, nor is it simply SE done poorly. Instead, it represents an alternative community-intensive socio-technical approach to developing software systems, artifacts, and social relationships. However, it is not without its limitations and constraints:

First, an individual developer's interest, motivation, and commitment to a project is dynamic and not indefinite [Robles and Gonzalez-Baharona 2006]. Some form of reciprocity and intrinsic motivation seems necessary to sustain participation, and a perception of exploitation by others can quickly dissolve a participant's commitment; worse, it may lead a participant to persuade others to abandon a project that has gone astray. Nonetheless, the fact that large numbers of individuals globally distributed do contribute to FOSSD projects suggests that the challenge is to mobilize and direct action towards collectively desirable outcomes [cf. Dutton 2008].
What are the most effective ways to start new FOSSD projects so as to enable the easy recruitment and participation of new/experienced FOSS developers?

- What are the most effective incentives for motivating FOSS end-users to become project contributors, developers, or core developers?

Second, FOSSD projects do not escape conflicts in technical decision-making, cooperation, coordination, or control. As these projects generally lack traditional project managers, they must become self-reliant in their ability to mitigate and resolve outstanding conflicts and disagreements. Values that shape system design choices, as well as choices over which software tools to use, and which artifacts to produce or use, are determined through negotiation rather than administrative assignment. Negotiation and conflict management then become part of the cost that FOSS developers must bear. Time and effort spent in negotiation and conflict management represent an investment in building and sustaining a socio-technical network of dependencies. Furthermore, it may be that the success of FOSSD projects primarily depends on the social networks that emerge within and across projects, and perhaps across different software ecosystems. A number of recent studies [Ducheneaut 2005; Madey, Freeh, *et al*. 2005; Toral, Martinez-Torres, *et al.* 2010; Sowe, Stamelos, *et al.* 2006] have discovered and modeled how participants in these social networks are recurrently structured around small numbers of key project contributors, who tend not to be core developers or end-users, but who serve as "knowledge brokers" or "linchpin developers" continually bringing together the otherwise loose-knit community of practice that situates a FOSS system project. Exhibits 6 and 7 provide examples of these small worlds.

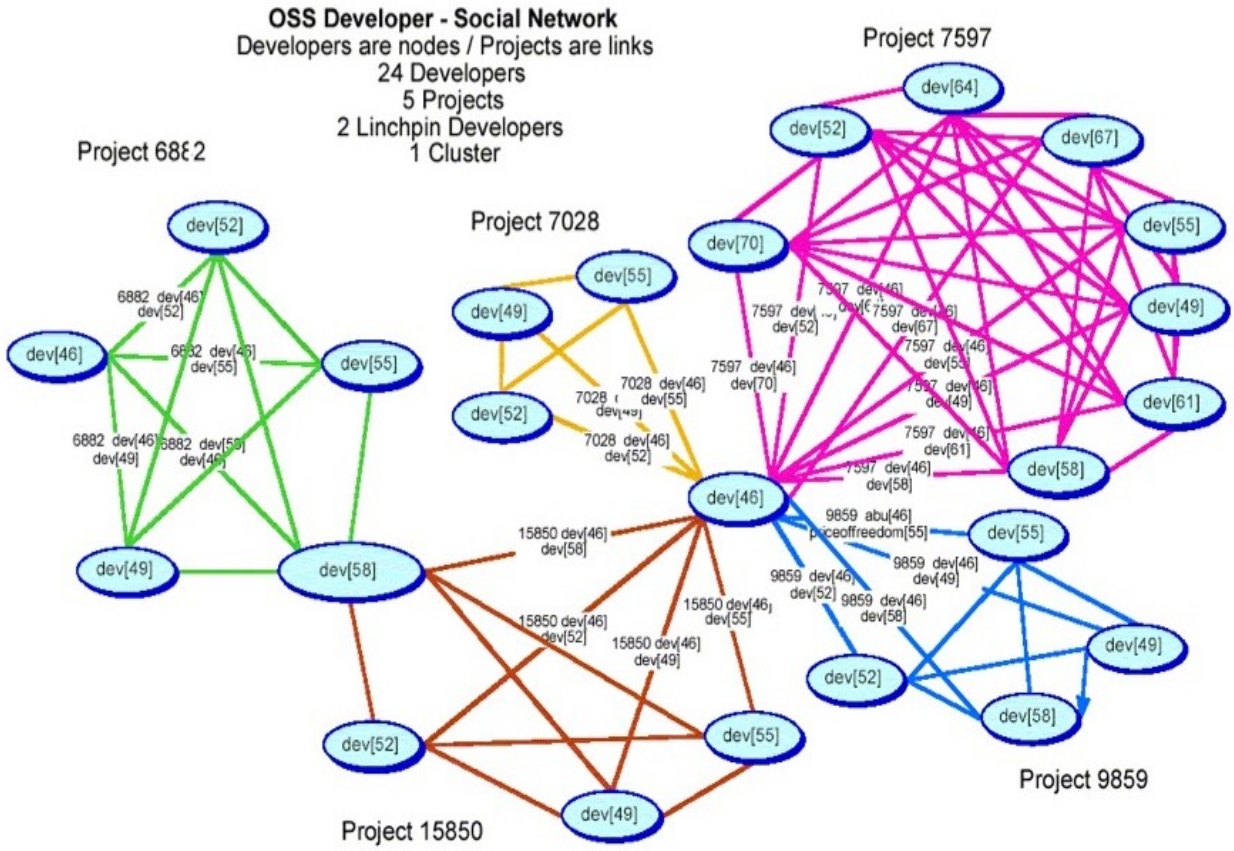


Exhibit 6: A social network of FOSS contributors spanning five projects that are interlinked through two linchpin developers [Madey, Freeh, *et al*. 2005].

Social networks are becoming a significant resource for enabling FOSSD project contribution and sustained success, yet the body of knowledge and process maturity models says little or nothing about the value of software developer networks.

- What are the roles of social networks in facilitating or inhibiting the development of large software systems in general, and FOSS systems in particular?
- What is the value of social network modeling and visualization tools in improving the self-organization of FOSSD projects?
- Under what conditions does the growth of a social network of FOSS developers correspond to the exponential growth and evolution of FOSS system size, functionality, and capability?

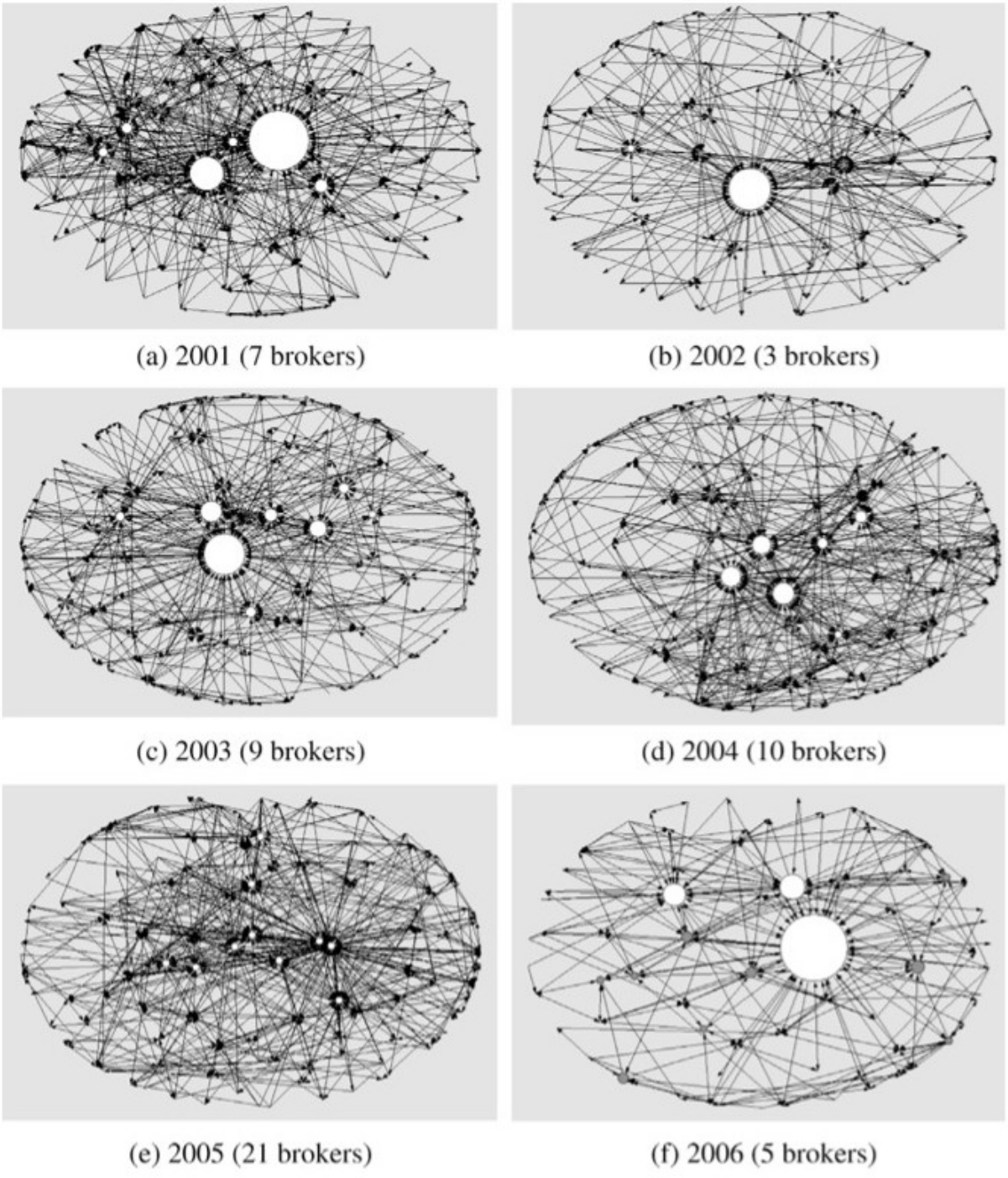


Exhibit 7: Social networks of FOSS developers in one project (the ARM Linux Community), highlighting the emergence of different knowledge brokers (white spots) over five years [Toral, Martinez-Torres, *et al.* 2010].

Third, there is no single FOSSD process. If one wants to join, learn from, or emulate the success of a FOSSD project, there is no single or reference project to consider. Why? To begin, FOSSD projects vary in size: from small efforts involving 1-2 people (most of which fail), to large community project efforts with core developers and numerous contributors and users, to large foundation-based projects affiliated with a non-profit corporation that provides some governance and legal protection. Project team size likely matters, but how it matters is unclear. Similarly, FOSSD project location(s) matter: small projects may involve individual or collocated developers, or they may involve developers who are geographically dispersed; large projects are generally multi-site and spread across time zones, and some are spread across national and cultural boundaries; FOSSD projects in non-English speaking and writing cultures are inaccessible and invisible to English-only developers; commercial efforts may include some or all of these. Once again, location matters, but how is unclear.

- Under what conditions will FOSSD projects become the preferred mode for developing large software systems that are intended for international markets or global applications?

- How are FOSSD projects in non-English speaking and writing cultures similar to and different from those in English cultures?

- How does FOSSD project size, complexity, and practice vary across cultures?

Fourth, alliance and community building through participation, artifacts, and tools points to a growing dependence on other FOSSD projects. The emergence of non-profit foundations that were established to protect the property rights of large multi-component FOSSD projects creates a demand to sustain and protect such foundations. If a foundation becomes too bureaucratic, this may drive contributors away from its FOSSD projects. These foundations need to stay lean and not become a source of occupational careers, in order to survive and evolve. Similarly, as FOSSD projects give rise to new requirements for community building, community software, and community information sharing systems, these requirements need to be addressed and managed by FOSSD project contributors. FOSSD alliances and communities depend on a rich and growing web of socio-technical relations. If such a web comes apart, or if the new requirements cannot be embraced and satisfied, then the FOSSD project community and its alliances will begin to collapse.

- When are multi-project alliances more effective than non-profit foundations at advancing the collective interests and governance of related FOSSD projects?

- How can SE processes and practices be adapted to facilitate multi-project system development alliances?

- How are project community requirements elicited, analyzed, traced, and validated? How do such requirements evolve over time?

Fifth, individual and shared resources of people's time, effort, skill, values, and computing resources are part of the socio-technical web of FOSSD. Existing software systems are reinvented as FOSS because communities emerge that seek to make such reinvention occur. FOSS systems represent a shared commons of reusable and adaptable resources that require collective action. Without this collective action, the common pool will dry up. Without the common pool, the community fragments and disappears.

- What are the many forms and practices for reusing FOSS system artifacts within and across FOSSD projects?

- What are the development (within release) and evolutionary (across releases) life cycle of FOSSD artifacts, and how do their life cycles vary by FOSSD project size, location, and application domain?

- Under what conditions will successful, high growth FOSS systems transition into slow or no growth systems?

- Under what conditions will software ecosystems emerge which facilitate the merger or alliance of small FOSSD projects in order to become larger, successful, and high-growth project communities?

Sixth, empirical studies of FOSSD are expanding the scope of what we can learn about the development of large software systems. In addition to traditional methods used to investigate FOSSD like reflective practice, industry polls, survey research, and ethnographic studies, comparatively new techniques for mining software repositories [Bajracharya, Ossher, *et al.* 2009; Howison, Conklin, *et al*. 2006; Ossher, Bajracharya, *et al*. 2009] and multi-modal modeling and analysis of the socio-technical processes and networks found in sustained FOSSD projects [Sack, Detienne, *et al.* 2006; Scacchi, Jensen, *et al.* 2006] show that the empirical study of FOSSD is growing and expanding. This in turn will advance empirical science in fields like SE, previously limited by the restricted access to data characterizing large, proprietary software development projects. Thus, the future of empirical studies of software development practices, processes, and projects will increasingly be cast as studies of FOSSD efforts.

- What methods for comparative study of large samples of FOSS systems and development projects provide the insight and rich contextual descriptions of ethnographic field studies, along with the scalability of automated data mining and discovery tools?

Additional Research Opportunities for FOSSD and SE

A significant number of opportunities and challenges arise when we identify software developments or socio-technical interaction practices in FOSSD projects that might be applied in the world of SE. Let us consider some research opportunities for SE that can arise from FOSSD studies.

FOSSD is poised to fundamentally alter the costs and constraints of accessing, analyzing, and sharing software process and product data, metrics, and data collection instruments, and will thus have a profound impact on SE. [Cook, Votta, *et.al.* 1998; Harrison 2001; Scacchi 2006]. For example, software process discovery, modeling, and simulation research [Scacchi, Jensen, *et al*. 2006] is one arena in which lower costs can be advantageous. Similarly, the ability to extract or data mine software product content (source code, development artifacts, team communications, public user feedback) within or across FOSSD project repositories [Howison, Conklin, *et al.* 2006] to support its visualization, refactoring, or redesign can be a high-yield, high impact area for SE study and experimentation. Another would be examining the effectiveness and efficiency of traditional face-to-face-to-artifact SE approaches or processes for software inspections [Seaman and Basili 1998] compared to the online informal peer reviews involving “many eyeballs” prevalent in FOSSD efforts.

Studies of motivation, participation, role migration, and turnover of individual FOSS developers, suggest that the SE community would benefit from empirical studies that examine similar conditions and circumstances in conventional software development enterprises. Current SE textbooks and development processes seem to assume that individual developers have technical roles and motivations driven by financial compensation, technical challenge, and the quality assuring rigor that purportedly follows from the use of formal notations and analytical schemes. Said simply, is FOSSD more fun, more interesting, more convivial, and more personally rewarding than SE, and if so, what can be done to make SE more like FOSSD?

Studies of resources and capabilities employed to support FOSSD projects indicate that conventional software cost estimation or accounting techniques (e.g., “total cost of operation” or TCO) are limited to analyzing resources or capabilities that are easily quantified or monetized. Social and organizational resources are slighted or ignored by

such techniques, producing results that miscalculate the diversity of resources that affect the costs of software development projects, whether FOSS or SE based.

Studies of cooperation, coordination, and control in FOSSD projects indicate that virtual project management and meritocratic socio-technical role migration and advancement can provide a lighter-weight approach to SE project management. It is unclear whether SE corporate efforts will eschew traditional project management and administrative control in favor of the freedom of choice and expression that may be necessary to provide the intrinsic motivation to self-organize and self-manage SE projects.

The results of studies of alliance formation, inter-project social networking, community development, and multi-project software ecosystems, suggest that SE projects currently operate at a disadvantage compared to FOSSD projects. In SE projects, it is commonly assumed that developers and end-users are distinct communities, and that software evolution is governed by market imperatives, the need to extract maximum marginal gains (profit), and resource-limited software maintenance effort. SE efforts are set up to produce systems whose growth and evolution is limited, rather than capable of sustaining exponential growth of co-evolving software functional capability and developer-user community seen in successful FOSSD projects [Scacchi 2006].

From studies of FOSS as a social movement [Elliott and Kraemer 2008; Elliott and Scacchi 2008], it appears that there is an opportunity and challenge for encouraging the emergence of a social movement that combines the best practices of FOSSD and SE. The world of open source software engineering (OSSE) is the likely locus of collective action that might enable such a movement to arise. For example, the community Web portal for Tigris.org (http://www.tigris.org) is focused on cultivating and nurturing the emerging OSSE community (see Exhibit 1 above). More than 700 OSSE projects are currently affiliated with this portal and community. It might therefore prove fruitful to closely examine different samples of OSSE projects at Tigris.org to see which SE tools, techniques, and concepts are being employed, and to what ends, in different FOSSD projects.

Questions regarding the quality, security, productivity, reliability, architecture, and governance of FOSS systems or projects are still frequent, and research results vary, sometimes finding positive, negative, or no relationships when compared to systems whose development was guided by SE. How can this be? First, such results will vary depending on which software systems and development settings are examined, as not all systems or settings are directly comparable, nor are the methods for how they were created; at least, not without careful sampling and empirical research design. Second, if SE and FOSSD are alternative approaches to developing large software systems, is “software quality” just an attribute of a body of source code (e.g., ratio of detected errors per thousand lines of code)? Or [cf. Rusovan, Powell, *et al.* 2005], is it perceived as the

ease with which users access and update the source code to overcome a perceived flaw? What is and is not quality software depends on its definition and application. Again, it depends on which systems and settings are being examined. Our scientific knowledge of SE and FOSSD is weak and too often anecdotal in these areas. Third, we have learned that in FOSSD projects and SE projects, the social or community architecture of a project is related to the technological architecture of the functional software produced by the project. But this relationship is not simple, nor readily predictable from external form or internal interdependency of one versus the other, nor is it necessarily persistent or controllable over time [cf. MacCormack, Baldwin, *et al.* 2010]. Yet there is still great interest in determining when they will be optimal, and therefore can be prescribed. This is a daunting problem that seems to merit not just further study, but alternative characterizations, reformulations, and visual renderings as another way to gain the insights we seek. Finally, when contrasted, both SE and FOSSD processes, practices, and project forms can help reveal where unquestioned assumptions or definitions lie, and where new formulations that are neither just social, nor just technological, but are jointly socio-technical may provide new kinds of insights and knowledge for how to develop large software-intensive systems.

Conclusions

FOSSD is emerging as an alternative approach for developing large software systems. FOSSD employs socio-technical work practices, development processes, and community network project forms often different from those found in industrial software projects, and those portrayed in software engineering textbooks [Sommerville 2006]. As a result, FOSSD offers new types of practices, processes, and organizational forms to discover, observe, analyze, model, and simulate, as well as to explain, predict and control. Similarly, understanding and explaining how FOSSD practices, processes, and projects are similar to or different from their traditional SE counterparts is an area ripe for further research and comparative study. Many new research opportunities exist in the empirical examination, modeling, and simulation of FOSSD activities, efforts, and communities. Furthermore, we cannot rationalize or predict when, where, why, how, or with whom FOSSD projects will work effectively or efficiently. Similarly, we lack the scientific knowledge needed to explain how FOSS systems evolve over time as well as within or across different software ecosystems. Nonetheless, the popularity and unbridled enthusiasm of thousands of young software developers participating in and contributing to FOSSD projects indicates that FOSSD processes, practices, and projects are being diffused, adopted, and adapted (transferred) in ways that we lack the scientific knowledge to understand.

FOSSD project source code, artifacts, and online repositories represent new publicly available data sources of a size, diversity, and complexity not previously available for

SE research, on a global basis. For example, software process modeling and simulation research and application has traditionally relied on an empirical basis in real-world processes for analysis and validation. However, such data has often been scarce, costly to acquire, and is often not available for sharing or independent re-analysis for reasons including confidentiality or non-disclosure agreements. FOSSD projects and project artifact repositories contain process data and product artifacts that can be collected, analyzed and shared in a free and open source manner.

Through surveys of empirical studies of FOSSD projects [e.g., Aksulu and Wade 2010, Crowston, Wei, *et al.* 2010; Hauge, Ayala, *et al.* 2010; Scacchi 2007], it should be clear there is an exciting variety and diversity of opportunities for new research into FOSS systems, development processes, work practices, project/community dynamics, and related socio-technical interaction networks. Thus, research going forward should engage more studies of SE *in practice* in everyday real-world settings, or apply FOSSD concepts, techniques or tools that can be collectively advanced through empirical studies that examine FOSSD processes, practices, and projects.

Collaboration

Overview

At its core, FOSS is about collaboration.

Free and open source software (FOSS) is transforming not only technology, but is influencing the way government operates (e.g., Obama's Open Government Initiative), and is expanding into other collaborative domains (e.g., Wikipedia, Open Courseware Consortium, etc.). Part of that transformation lies in new and novel ways of collaborating. FOSS has demonstrated collaboration on a very large scale, using poorly or undefined organizational structures, and un(der)-specified work methodologies. How do FOSS projects collaborate? What can we learn from this model of collaboration? How is FOSS collaboration changing? How is FOSS transforming the way the world collaborates? Lastly, how and what factors are influencing this direction?

FOSS describes a broad body of computing technologies. It is also a term that describes a set of collaborative principles. These principles have been used in the context of software development for over twenty years. Collaboration means people working together to accomplish shared goals. But emergent technologies support, and in some respects participate in collaborative processes (e.g., RSS feeds, open data, web services). Because technology in the information age offers the ability to collaborate over distance and time, collaboration is inherently a socio-technical phenomenon occurring on multiple spatial (local-to-global), temporal and human scales. Collaboration used to involve more face-to-face interaction between humans. Now it happens on a global scale, and thanks to the cyberinfrastructure that blankets most of the Earth, forms of collaboration are emerging in many areas that are grounded upon Internet technologies.

FOSS, conceptually, is about collaboration. Programmers and others in the computing industry were the first to take advantage of computer networks for collaboration. One of the great foundational innovations that led to the emergence of FOSS as an important socio-technical phenomenon — the creative use of copyright or FOSS licensing to encourage sharing and collaboration (e.g., General Public License and other FOSS licenses) — was developed or inspired by programmers. Given this history and current state, understanding FOSS as a collaborative paradigm is of vital importance for grasping what is happening in the political economy of the software industry and how FOSS-like practices are being "ported" or transferred into other areas beyond software development. Important examples include open content creation and peer-production

systems (e.g., Wikis, educational content sharing, photo and video sharing, game modding, music and other digital content remixing). Open source collaborative practices are shaking up major sectors of the global economy (e.g., journalism, publishing, the music industry). FOSS-like collaboration has potentially far reaching and transformative effects.

It is well understood that FOSS projects are *social worlds* [e.g., Strauss 1978, Gerson 1983, Star and Ruhleder 1996, Scacchi 2008]; that is, collective activities organized around a common topic or subject-matter that utilize shared communication channels. Various research perspectives use different terminologies and approaches to understanding FOSS projects as social and collaborative phenomena. For example, Benkler [2006] refers to FOSS development as the quintessential example of "commons-based peer production," while others like English and Schweik [2007] examine FOSS using a "commons" analytical framework. Kelty [2008], looks at FOSS collaborations with a broader perspective, labeling them "recursive publics," where there exists some common knowledge, and people can participate in and make modifications to that body of knowledge. Howison [2009] sees FOSS projects as "layered collaborations" where individual developers undertake short tasks that build upon a common software code base, which in turn enable others to work together or to take advantage of such work in progress. Still others view FOSS projects as institutional endeavors that can take the form of virtual organizations or be affiliated with non-profit foundations or for-profit corporations. These are just a view vantage points of FOSS. All have collaboration, to some degree, as a common theme. Moreover, throughout discussions in this workshop, participants were keenly aware of the ability to take collaborative principles found initially in FOSS projects (such as the innovation of FOSS-licenses to promote or permit collaboration) and extending them to other domains. Of interest here is how studies of FOSS collaboration can inform both future FOSS-based endeavors as well as the broader realm of "open content" collaboration. We see FOSS as an exemplar of a social production community with a (relatively) long track record. As such, knowledge and collaborative approaches generated within FOSS communities have the potential to be generalized to other social production communities.

Collaboration in FOSS occurs over multiple layers, scales, and settings. Thinking about FOSS in scales help to organize collective thinking about important research questions (see the "Outstanding or Emerging Research Problems" section below). Understanding FOSS as collaboration involves focusing attention on multiple levels or scales: (1) individual collaborators; (2) individual projects; (3) software ecosystem; and (4) regional and global issues.

The individual working in a group, team, alliance, or coalition spanning multiple social worlds can be viewed as the building block or unit of analysis in FOSS collaborations.

Projects are the most obvious settings where collaboration occurs. As many FOSS projects are not collocated. With geographically dispersed developers and users, collaboration happens online and through shared FOSS artifacts. At this level, collaboration involves “core-contributions” by software developers as well as interactions between developers and the users of their software.

Ecosystem-level issues involve, for example, the dynamics that may exist when projects involve not just volunteer developers but also include the interests of firms, nonprofits (e.g., foundations), and actors from government agencies. In addition, software ecosystems more fully articulate the software supply networks that associate software producers, integrators, and consumers [Jansen, Brinkkemper, *et al.* 2009].

Regional level collaboration engages issues of culture, language, social movements and other heterogeneous components, while global collaboration concerns broad political-economic or national policy-related issues. For example, governments outside the US are considering FOSS as an integral component of their IT procurement policy, sometimes as a method for building or supporting their in-country software industry [Lewis, 2010]. From a collaboration standpoint, FOSS is seen as a way to act collectively at a national scale to bolster in-country IT industries. Governments are also seeing FOSS as a potential avenue toward interoperability between and across government levels — potential systems that help to remove the standard “stovepiping” of information (consider, for example, the potential relationship between FOSS and the Obama Administration's Open Government Initiative in the U.S.). Governments are seeing opportunities to work collaboratively with constituents in two-way information flow systems — e.g., crowd sourcing, the use of FOSS by citizens to create mashups or other information flows rapidly when needed (e.g., Katrina, Haiti). In addition, some in industry see FOSS and its collaboration principles as a business model to gain competitive advantage. Others see FOSS as a threat to their industry and livelihood. Again, the openness of FOSS is being ported to and affecting other arenas.

Observation and Intervention

Research into FOSS traditionally breaks down into two different approaches: the “descriptive observation” approach, and the “prescriptive (or proscriptive) intervention” approach. Social science researchers are frequently interested in how FOSS systems develop and grow, and the interactions between participants in these systems. Such researchers traditionally operate under the idea of “look but don’t touch.” They are willing to communicate what they find but are hesitant to inject the results of their observations into the specific communities they observe, or to impact these

communities in the course of their research. The unique collaborative environment of FOSS projects provides a fertile ground for this kind of social science research.

Conversely, engineering researchers are most interested in how they can (positively) impact practice in the environments they study. FOSS systems and communities are living, functioning entities, and the results of research into those communities can be used to improve them. As proprietary and FOSS development methodologies converge, the results of this research will also be applicable for commercial software development.

We feel that both approaches are useful for the future of research in FOSS as a new and important paradigm for how humans collaborate across geographic scales in software and in other domains.

Research Findings

As suggested above, a valuable approach to analyzing FOSS as a collaborative phenomenon is to look at it across multiple scales or levels: individual, project, ecosystem, and regional/global. In recent years there has been a large body of research that could be classified as “FOSS as collaboration,”focusing on one or more of these levels of analysis. This section provides a sample of such research.

Collaboration among individual FOSS developers

The largest body of research on FOSS has focused on the individual scale, investigating why FOSS developers — particularly volunteer developers — contribute to FOSS projects. Examples of this stream of research include (but are by no means limited to) Ghosh [2005]; Lakhani and Wolf [2005]; and Chakravarty, Haruvy, *et al*. [2007]. The idea that volunteer programmers would freely contribute their intellectual property puzzled, in particular, economists (see, for example, Lerner and Tirole [2005a]). From such studies volunteer motivations become clear: the opportunity to learn, signalizing to peers one’s abilities, enjoyment, filling a (software) need, and, to some degree, helping to sustain the FOSS movement [Elliott and Kraemer 2008].

However it has recently become apparent that collaboration in FOSS has become more complex, involving not just volunteers but also paid participants from firms, government agencies and nonprofit foundations who may provide legal, financing, marketing, and collaborative infrastructure support [Schweik and Kitsing, 2010]. Consequently, the incentives for participation for individual developers are becoming more complex, as are the development settings in which they operate [Jensen and Scacchi 2010]. Results

show that individuals' motives, as well as their occupational and career contingencies [Elliott and Scacchi 2003, 2008; Jensen and Scacchi 2007] are not static, but evolve over time. By studying OpenOffice.org, Freeman [2007] argued that individuals' motivations to join and continue to participate in the FLOSS projects are related to personal history prior to and during participation. In the *PhpMyAdmin* project, Fang and Neufeld [2009] revealed that initial motivations to participate do not effectively predict long-term participation, while situated learning and identity construction behaviors are positively linked to sustained participation. Iivari [2008], p. 512, describes the role of users (not developers but non-technical, non-computer professional users who are not interested in OSS development, but only in the resulting solutions) in the OSS.

Collaboration among FOSS projects

At the project level, Mockus, Fielding, *et al*. [2002] provided an early analysis of how Open Source projects and their associated communities interact, as suggested by an investigation into the Apache Software Foundation and the Mozilla project. The authors discovered significant inequity in work performed: a core group of individuals contributed the majority of the programming effort and hundreds of others provided only very small contributions. This suggests that rather than trying to determine whether there is some magic number of core developers needed for a FOSS project to succeed, it is important to create and sustain a critical mass of FOSS developers who configure their development practices to provide the socio-technical direction, decision-making, and governance actions that keep the project moving forward.

Drawing on social network theories and previous studies, research on collaboration among FOSS projects also examines the dynamics of social network structures in FOSS teams. For example, studies like Long and Siau [2007] suggest that the interaction pattern of a FOSS project evolves from a single hub at the beginning to a core/periphery model as the project moves forward. Other studies like Madey, Freeh, *et al.* [2002, 2005], Jensen and Scacchi [2005], De Souza, Quirk, *et al.* [2007], and Toral, Martinez-Torres, *et al.* [2010] find that complex intertwined networks or socio-technical webs better characterize the structural patterns of collaboration within and between multiple, interrelated FOSS projects.

A variety of case studies have emerged since the Mockus, Fielding, *et al*. [2002] study, many focusing on large (in terms of developers and user communities), high profile FOSS projects. For example, O'Mahony and Ferraro [2007] analyzed the evolution of governance in the Debian Linux project. And recently more attention has been paid to how to analyze and structure FOSS project governance and institutions (e.g., [Schweik 2005; Markus 2007; O'Mahony 2007; Schweik and English 2007; O'Neil 2009]).

Other studies investigate the multiple dimensions of FOSS "success" and "failure" [Crowston *et al.*, 2003; Robles *et al.*, 2003; Weiss, 2005; English and Schweik, 2007a; and Wiggins and Crowston, 2010], and provide a foundation for an effort to identify factors that lead to these collaborative outcomes [Schweik *et al.*, 2010].

Collaboration among multi-project FOSS ecosystems

If we look at multi-project FOSS ecosystems, we find configurations such as project federations. Examples of such federations include the Apache Software Foundation, the Free Software Foundation, or even the *SourceForge.net* [2010] website. They typically consist of multiple projects with a shared culture and technical infrastructure, although their domains may be orthogonal. Jensen and Scacchi [2005] examine recurring collaborations in the context of work processes, looking at objects of interaction (or boundary objects) as a way of identifying evidence of interaction and collaboration between loosely-coupled projects within a FOSS ecosystem. Schweik and Kitsing [2010] provide a case study of a federation of FOSS projects developing geospatial software systems that are formally associated through a nonprofit foundation, and investigate how this federation affects project governance and operations.

Taking a slightly different view of FOSS ecosystems, Madey, Freeh, *et al.* [2002] examine multi-project collaborative networks of projects hosted on *SourceForge.net* [2010], identifying "linchpin" developers with membership in multiple projects. Such developers may play a similar role to "gatekeepers" in organizational studies, facilitating the flow of information and collaboration between projects. Such linchpin developers may play an even more significant role in multi-project federations, such as the Apache Software Foundation and the Free Software Foundation, where federated projects may share more than simply beliefs, values, and social norms. Consequently, linchpin developers may help to enable the critical mass of software developers, socio-technical actions, meritocratic coordination, and lightweight governance that span and sustain the larger web of FOSS projects in a software ecosystem.

Collaboration on a regional government or global scale

Governments around the globe have considered the issue of providing direct or indirect support to FOSS activities, turning FOSS into a political issue [Rossi 2006]. FOSS is seen to hold potential to promote technology neutrality and, additionally, to provide market regulation or freedom from "vendor lock-in" when competition is limited by one or a few dominant software companies [Lee 2006].

FOSS is a subject of interest within governments. Attention has centered on many issues, including security concerns arising from failing faith in the “security by obscurity” doctrine of software security [Schryen and Kadura 2009; Hoepman and Jacobs 2007]; reducing acquisition and maintenance costs; and increasing support for open standards necessary for long-term system evolution, document management and accessibility [MITRE 2003, Wennergren 2009, Wheeler 2009]. Such accessibility is a requirement for collaboration with other organizations, both within and outside of government. Additionally, FOSS has been advanced as a means of increasing democratic legitimacy through participatory models of administration, providing increased transparency in governance [Citron 2008]. Such participatory models leverage the collective intelligence of a population to produce information goods that outperform concentrated, authoritative efforts [Chadwick 2009]. Participatory governance models inspired by FOSS system methodologies, processes, and practices indicate a shift in governance towards project-centric collaboration between public officials and their constituencies, or with other non-governmental organizations.

Outstanding or Emerging Research Problems

In this section we will continue to organize our discussion around the analytic levels or scales discussed previously: (1) Individual contributors; (2) Projects; (3) Ecosystems; (4) Regional, Global or “Open” questions and (5) Cross-cutting concerns, though we will leave ecosystem and regional/global issues for later in this report.

Contributor-Level Collaboration

Individual developers are not collaboration systems in and of themselves, but they are the smallest social unit in FOSS collaborations. Contributors to FOSS include end-users, defect submitters and feature requesters, casual and core developers, project management committee members, release managers, community managers, foundation board members and leaders spanning multiple levels of involvement.

As we noted above, much of the early research on FOSS focused on individual (usually volunteer) motivations for participation. However one set of questions that is now emerging is how individual motivations are changing in more hybrid collaborative environments (see Project-Level Questions section, below), and whether and how individual behavior in FOSS collaborative environments differs across cultures.

In addition, various tools including discussion forums, email, bug trackers, IRC chat, and other social media electronically support distributed collaboration. Given that in many

cases these individual-level contributions (e.g., code) and individual-posted communications (e.g., forum posts, etc.) are all captured and stored within these systems, there are new, unprecedented opportunities to track and analyze individual developer behavior, and potentially, their "nano-scale" actions, and measure and assess drivers of individual productivity.

Potential research questions at the "Contributor" level include:

- *Developer Behavior*: How does the behavior of a contributor change over time as he or she gains skills, knowledge and authority in a FOSS development project? What kinds of situations lead him or her to leave the development effort? What kinds of artifacts, constructs, or affordances (e.g., tools, processes and practices, beliefs, social structures, technical structures) mediate the choices that developers make [Baldwin and Clark, 2005; Elliott and Scacchi 2003, 2008; Howison 2009; Scacchi 2010a]?

- *Developer Action Steps*: If we can track the discrete action steps of an individual developer how can we improve the productivity of the developer and the reliability of the code?

- *Behavior Across Cultures*: Do contributor behaviors differ across geographic regions or cultures (comparative studies of individual FOSS developers)?

Project-Level Questions

In this section we discuss the research questions that can be studied most directly at the level of an individual FOSS project. Most of these questions deal with the functioning of groups of FOSS programmers working together to create software. They explore the ways in which teams might be organized, the effect of that organization on the performance of the team, and the creation of software tools to improve the functioning of the team. Some research has already been focused on this set of questions, and there are some emerging results. However, in recent years there has been dramatic growth in projects that involve or are supported by for-profit corporations, nonprofit foundations or government agencies, rather than the "typical" FOSS project that historically may have relied on volunteer or unpaid developers. We focus here on questions that have not yet been completely explored, though there are partial results for many of these questions.

One of the opportunities at the Project Level is to explore the mostly "dead data" studies that have been done, translate them into process or tool interventions, and perform lab or field studies of these interventions. At the project level of analysis, at least three focal

areas can be addressed: 1) Collaborative Structure and Processes; 2) New Users, and 3)Collaborative Infrastructure.

Collaborative structure and process

- *Collaborative Activities:* What are the main collaborative activities and what socio-technical tools can best support these activities? Much research has looked at one or two projects to understand how collaboration is supported. But more generally, how are the existing tools used (or not used) to facilitate or inhibit collaboration? What underlying "operational level rules" are embedded in these tools (e.g., Lawrence Lessig's "Code as Law")? How can projects share tools? To what extent could these tools transfer for use in other FOSS-like collaborative processes?

- *Project Scale*: FOSS projects number in participants from one (or none, in the case of abandoned projects) to hundreds of thousands. What enables collaboration on such divergent scales? What inhibits collaboration? (How) do collaboration structures and processes change as project populations grow and shrink?

- *Project Governance and Institutions*: How do FOSS "institutions" (norms, operational procedures, more formalized rules, project management and governance) evolve over time? How do they vary based on team composition (e.g., all volunteer, all paid, hybrid teams)? Do foundations and businesses influence institutional change? How so?

- *Variation in Collaborative Practice*: How do collaboration practices differ among different types of projects? Across different cultures? Does FOSS collaboration differ between projects that are more "geographically homogeneous" (members all from one country, for example), compared to teams made up of participants from multiple countries or world regions?

- *Collaboration Initiation and Life Span*: How are teams formed? Why do they form in specific ways? Are there recurring patterns in team formation across projects, foundations, etc., and how can they be facilitated? What is the life span and trajectory of a team? Can we identify team life cycles?

- *Encouragement:* How can we encourage and support contributions from user experience experts and what does this mean for successful collaboration across disciplines, i.e. developers, users, domain experts, and designers?

- *Collaborative Failure*: How do FOSS collaborations fail because of interdisciplinary differences, and how can tools and processes be developed to facilitate the resolution of problems?

- *Conflicts*: How are conflicts in FOSS teams resolved? How can they be resolved more effectively? What kinds of tools may help?

- *Developer/Community Relationship*: What is the relationship between user communities and development teams? Are they generally similar across projects? In what ways do they differ and how does that influence project direction and evolution?

New Users and Members

A key aspect of a successful community is its ability to bring in new users to replace those who leave, and to bring in new members to "grow" the project. All communities lose users eventually, so successful communities must be skilled at incorporating new users. There are many challenges in bringing in new users, including recruiting them, socializing them to group norms and practices, and helping them find work to do that fits their experience and interests. We include sample research questions in each of these areas below.

- *Membership Life Cycle and Management*: What are the life cycle trajectories of contributors participating in FOSS projects? Is this life cycle and corresponding life span ideal? How can sufficient users be brought in to refresh the membership? What is the best mix or configuration of characteristics of members for a project, in terms of the type and quantity of work they are able to take on? How does the configuration of members change over the life cycle of the project?

- *Member Recruitment*: How are new members (users, developers) "recruited" into FOSS projects? What methods are effective in managing different skill capabilities and mentoring new developers? Do new FOSS projects depend on the prior FOSS project success of their founding core developers?

- *Broader Interactions with User Community*: How is the broader user community supported and encouraged? What "marketing" approaches encourage the growth of a user community?

Collaborative Infrastructure

Other online tools and social networks can be instrumented to enhance the understanding of collaborative interactions among contributors to a given FOSS project [Star and Ruhleder 1996, Scacchi 2007]. The potential for deploying these tools to detect, collect, and visualize social network and technical configuration interdependencies seems increasingly plausible [De Souza, Quirk, *et al.* 2007], as popular interactive FOSS development environments like *Eclipse* and *NetBeans* include uploads of (anonymized) usage data. Further, research has begun to investigate instrumenting FOSS tools, for example user data collected in *GIMP* [Terry, Kay, *et al.* 2008]. Contributor-level collaborations are important for understanding and improving communication and productivity, and effective communication is foundational for successful collaboration. Communication among FOSS contributors occurs one-to-one, one-to-many, or many-to-many to share information and provide markers for awareness of activities in progress, completed, or abandoned. Artifact usage potentially enhances communication success. The existence of FOSS development artifacts [Ekbia 2009; Robles, Gonzalez-Barahona, *et al.* 2006; Scacchi 2002, 2010a] creates a great and unique opportunity to study communication patterns or discourse networks in rich detail.

- *Communication Networks and Structure*: What are the communication networks across different contribution types?

- *Passive Communication*: How are artifacts useful as a passive knowledge transfer tool? How do contributors learn from each other by studying the code and other artifacts created and shared through the FOSS repository?

- *Mentoring Tool Creation*: How can tools be created to help new members of a FOSS project find a mentor who can help them fit into the project and find ways to effectively contribute?

- *Increasing Participation*: How can tools be created that encourage existing members of a FOSS project to interact with new members of the project in ways that help new members feel welcomed?

- *Conflict Avoidance*: Can tools be created that help to protect against “stepping on others' toes?” For example, a tool that visualizes past collaboration on a FOSS project can help a contributor understand which collaborators will be affected by a change under consideration, allowing communication between collaborators before investing a lot of time in making the change.

Cross-Cutting Concerns

Lastly, there exists a variety of issues related to FOSS development that are cross-cutting across multiple levels of observation. These issues interact with the community and are shaped by it, causing cascading effects that make them difficult to study and understand within a purely hierarchical framework.

The architecture of the software and its relationship to communication structures within a software development project, first proposed by Conway [1968], is one example of such a cross-cutting concern. For example, a community that is built around a monolithic socio-technical structure, such as found in the *GNOME* project, requires intense involvement in core elements of the source code for even some minor projects. In comparison, a community with a modular socio-technical structure, such as in the *Eclipse* project, allows firms and individuals to work independently, minimizing coordination and collaboration needs. It is precisely this interaction of collaboration needs and architecture that causes this to be a cross-cutting concern.

Understanding the architecture of a FOSS community involves more than simply understanding how the code is structured. It cascades down to the actions of the individual who must write code, the project which needs to allocate work according to technological constraints, and foundations which are made up of projects and have their own architectures of coordination. These issues also affect other peer production and social content systems, such as Wikipedia, where the design of the tools and interactions of the community directly impact the degree to which the community can collaborate, attract new members, and create new content.

Related to architecture and cross-cutting concerns, sample research questions include:

- *Architecture*: How does the architecture of a FOSS system mediate the collaboration within and across teams, projects, or ecosystems [cf. Ovaska, Rossi, *et al.* 2003]? Can we provide guidance to teams designing or refactoring a project to create a more collaborative environment? Do different architectural configurations and/or development processes and practices lead to different forms of collaboration?

- *Governance*: To what degree does choice of project governance structure or foundational form (e.g., in the U.S., those enterprises conforming to civil/tax codes 501c3, 501c6, etc.) affect development of FOSS projects? How do governance structures in FOSS projects compare to those found in closed source or proprietary software development projects?

Conclusions

As a prime example of commons-based peer production, recursive public, and layered collaboration phenomenon, collaboration in FOSS projects is different from collaboration in software engineering and is transforming the way people collaborate in other domains. Initial studies show it is complex: multi-layered, multi-faceted, and evolving. Understanding how participants in FOSS projects collaborate is vital to understanding how collaboration can be supported and improved, how and why it is evolving, and the potential opportunities and consequences of its transfer.

Ecosystems

Overview

Software development doesn't happen in isolation. It takes place in a collaborative ecosystem of stakeholders, resources and tools and technologies that are marshaled, created, transformed, and consumed in the production and maintenance of software systems.

Understanding in the interaction between projects and their broader ecosystems is key to assessing the potential and limits of FOSS production. We use the ecosystem metaphor to capture the notion of interdependence, mutual support and competition amongst the projects, code and the environment in which these operate. Software ecosystems have arisen not simply from code reuse but from shared culture, common standards (e.g., data formats and communication protocols), community culture and bylaws, processes, and tools. The societal effects of free and open source software extend beyond the participants, to all members of society.

To fully understand the potential and limits of open production therefore requires a conceptualization of the external factors affecting FOSS projects and their relations with project outcomes. A wide range of factors may be relevant. Project outcomes are affected by the availability and interest of new project participants, who bring with them training, experience and connections, or take these to other projects. Developers coming from different backgrounds are likely to bring different levels of skills and knowledge. Projects depend on a variety of tools for production or collaboration; these are likely created by other projects, often FOSS themselves. Projects may be dependent on the output of other projects (discussed above under collaboration).

Over time, code reuse has grown from the method/function level to libraries and more recently frameworks and platforms. As a result, FOSS projects likely build on the products of numerous other projects, as well as commercial code, and may in turn be a platform for additional products. However, projects may also compete for the attention of developers, sponsors and users. Projects may interact with for-profit companies that fund projects or sell competitive or complementary products and services. Projects embody a range of production and coordination processes, which can be conflictive or integrative [Jensen and Scacchi 2005]. FOSS projects rely on a range of infrastructures for developing and sharing code and knowledge, such as forges and code repositories. The ecosystem can include various societal institutions that govern projects, such as software foundations or the Free Software Foundation. Projects may also be influenced

by government policies and practices, e.g., encouraging or frustrating the use of open source systems as well as legal frameworks, e.g., for licenses, copyright law or patents. Finally, projects are affected by the cultures and beliefs of developers and users, for example, about the importance or relevance of sharing.

FOSS projects and products are deeply affected by the ecosystems in which they operate, but they simultaneously change them, by developing new systems, providing training and educational opportunities and mechanisms for technology transfer among developed and developing countries, or by influencing the evolution of intellectual property regimes. The interaction of these dual processes of influence — environment affecting projects and vice versa — creates an exciting phenomenological environment in which to study the process of influence and change itself.

Characterizing Interaction Among Projects in a Software Ecosystem

We would like to understand how projects in an ecosystem interact. We know software development processes interact to communicate and coordinate the exchange data and control [Feiler and Humphrey 1993]. Process interaction has been studied within other fields, including networking [Simpson 2003] and distributed systems [Andrews 1991]. The types of process interaction studied in these fields assumes that the interacting processes have little variation or evolution over time, lending to interaction that is predictable. But what of FOSS processes? Do FOSS processes interact so as to *integrate* smoothly and in predictable ways, so as to reify interaction processes? Or does such interaction provide *conflict* that requires recovery or redesign in one or more of the interacting processes, for example, due to competing process goals, that affects resource availability and/or control exchange between processes? Can we identify specific types of integration and conflict, such as have been observed in other fields, and do these types of integration and conflict follow what other fields have observed?

Successful interaction between components of a software ecosystem depends on the relationships between the organizations of which it is comprised. For example, Bluedorn, Johnson, *et al. [*1994] identify six mechanisms of interorganizational interaction 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 inter-organizational relationships, we employ them to characterize software ecosystems. Unsurprisingly, the degree of coupling carries implications for the degree of process integration between these organizations. Similarly, 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.

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 plugin 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 1: Interorganizational synchronization and stabilization mechanisms (after Bluedorn, Johnson, *et al.* [1994])

Level	Example	Description
Common Culture	FOSS 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	FOSS Web repositories	Access to each other's data by business processes that operate independently
Coordination	Meta-communities, tool integration, plugin 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

Table 2: Levels of business process integration (cf. Alter [1999])

Together, these give us a basis for examining the types of processes we can expect to find in software ecosystems. Further, they provide insight into interacting members of the ecosystem (stakeholders), their relationships, and the motivations of these relationships. Identification of these stakeholders, relationships, and concerns requires analysis of the interprocess communication among projects in an ecosystem. We address this next.

Interprocess Communication Among Projects in a Software Ecosystem

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 processes as normal, perhaps with small accommodations. They also reinforce infrastructural processes since they do not require changes in the interactions between communities. If the degree of accommodation or adaptation becomes too great, it can precipitate *conflictive* communication between project communities. Conflict may occur due to changes in tools or technologies shared between them, or in contentious beliefs about 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].

Process Integration

Process integration can be direct and explicit. It can also be indirect and implicit, such as through common data standards. These standards can be viewed as *boundary objects* [Star 1990; Pawlowski, Robey, *et al.* 2000]. Boundary objects are those that inhabit and span several communities of practice, as well as satisfy the informational requirements of each community. Following Alter's [1999] classification, shared standards connote a low degree of process interaction between organizations in a software ecosystem. However, other boundary objects exist, as shown in Table 3. Among FOSS projects, boundary objects observed thus far include (a) shared beliefs and culture [Elliott and Scacchi 2003, 2008], (b) community infrastructure tools, such as FOSS defect repositories produced by other affiliated organizations, 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. 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.

Community Infrastructure	
Object Type	***Example***
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	
Object Type	***Example***
Product Infrastructure Tools	Plugins, 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

Table 3: Common boundary objects that span the World Wide Web [Jensen and Scacchi 2005].

While certain boundary objects indicate a degree of interaction between processes in FOSS ecosystems, it yet is unclear how this interaction plays out. As long as each member of the ecosystem adheres to these standards, they may choose to operate independently, following their individual processes as usual. However, an ecosystem 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 systems developed by otherwise independent FOSS projects help adapt to change. 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 to an existing standard, the other communities act to support it or choose to reject it. Likewise, defects in data representations or operations in one software system can cause breakdowns or necessitate workarounds by others in the ecosystem. We look at the causes and negotiations of these conflicts next.

Process Conflict

Process conflict can precipitate or follow from process breakdown, disarticulation, or disintegration. Conflictive activities arise often from organizations competing for market share and control of the technical direction of an ecosystem or market and shared technologies. It also arises from less belligerent activities, such as introducing a new version of a tool or database that requires massive effort to incorporate and that other organizations depend on. In these cases, the organization placed into conflict may simply choose to reject the revised tool or technology, possibly selecting a suitable replacement if the current one is no longer viable. This path was chosen by shareware/open source image editing projects 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 [Battilana 1995].

Conflicts across FOSS projects get resolved 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 (email, chat, instant messaging, etc.). Alternatively, an organization causing or resisting a tool or technology may succumb to pressure from the rest of the ecosystem. Irreconcilable differences, if they persist and are strongly supported, can lead to either unresolved conflicts (e.g., software updates that do not get implemented), incompatibilities in the interoperating software systems, or divisions in the ecosystem.

Synchronization and stabilization of shared artifacts, data representations, and operations or transactions are required for an ecosystem to be sustained. This process is not “owned,” [Larsen and Klischewski 2004] located within, or managed by a single organization or 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, approaches to modeling these processes often trade off representational detail of individual process forms, and instead uses a more abstract, low fidelity representation [Atkinson, Weeks, *et al.* 2004]. This is done only so as to 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 resource flow between interacting communities within an ecosystem, as suggested by Figure 1. Each channel between communities connotes *ad hoc* processes that articulate the interoperability or interdependence of tools and technologies, as well as the boundary objects, shared between them. The process characterizing the growth and evolution of a software ecosystem can therefore be characterized by the communication flow that enables integration or conflict process activities between constituent projects.

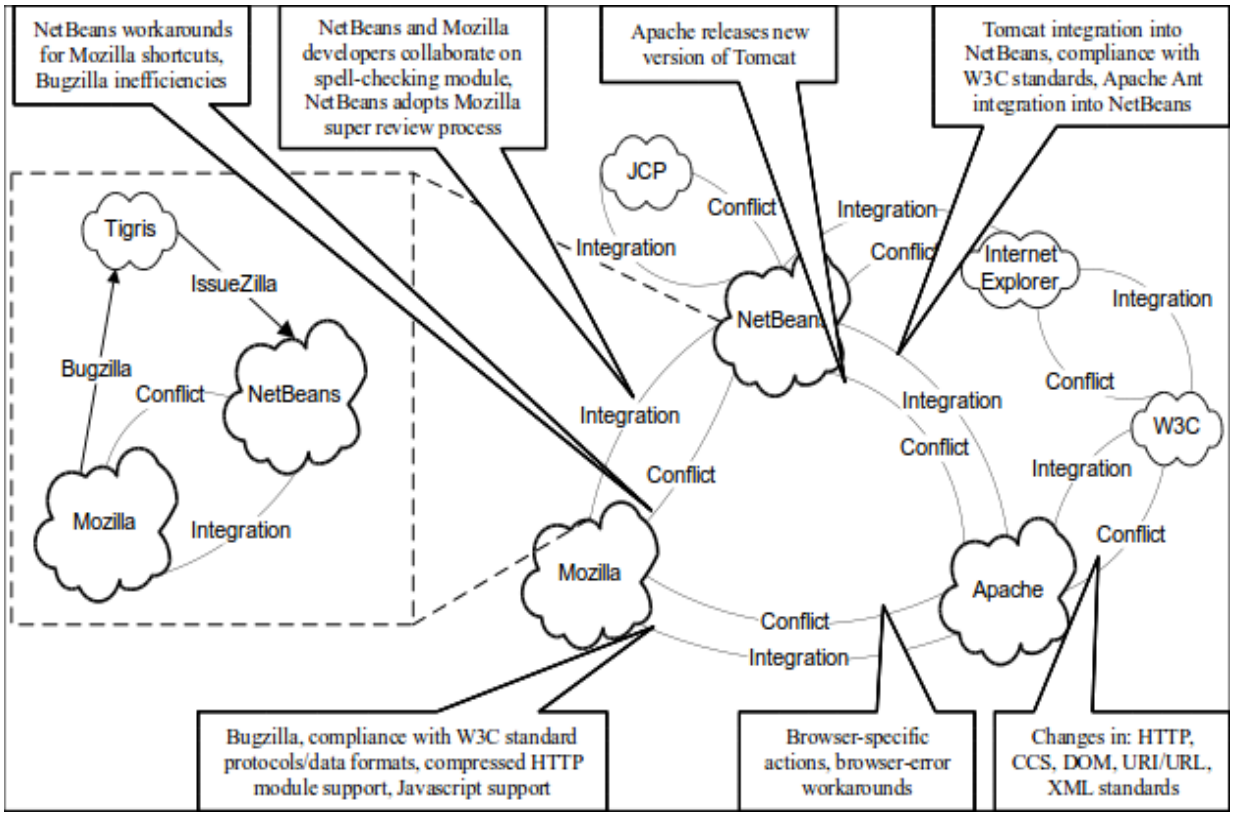


Figure 1: Process integration and conflict in a software ecosystem that spans the Web information infrastructure [Jensen and Scacchi 2005].

A Sample of FOSS Ecosystems

Here we briefly examine three different kinds of FOSS ecosystems. These are centered respectively on the development of FOSS systems for networked computer games, scientific computing for astrophysics, and World Wide Web. Along the way, example

software systems or projects are *highlighted* or identified via external reference/citation, which can be consulted for further review. One important concept to recognize is that each FOSS ecosystem represents a different mix of FOSS and non-FOSS producers, system integrators who may build systems from FOSS and non-FOSS components, and end-users who may seek alternative system configurations of an integrated (or loosely coupled) system. This set of relationships may be specified as a "software supply network" that interlinks FOSS producers, integrators, and consumers through the FOSS they share, modify and redistribute, as suggested in the following figure. This figure also highlights how the (copy)rights and obligations found in different FOSS licenses are composed or transferred before reaching end-users. So a FOSS ecosystem is not a single unitary object, but instead represents a web of interrelated participants, technologies, and other resources that interact across an underlying information infrastructure or cyberinfrastructure.

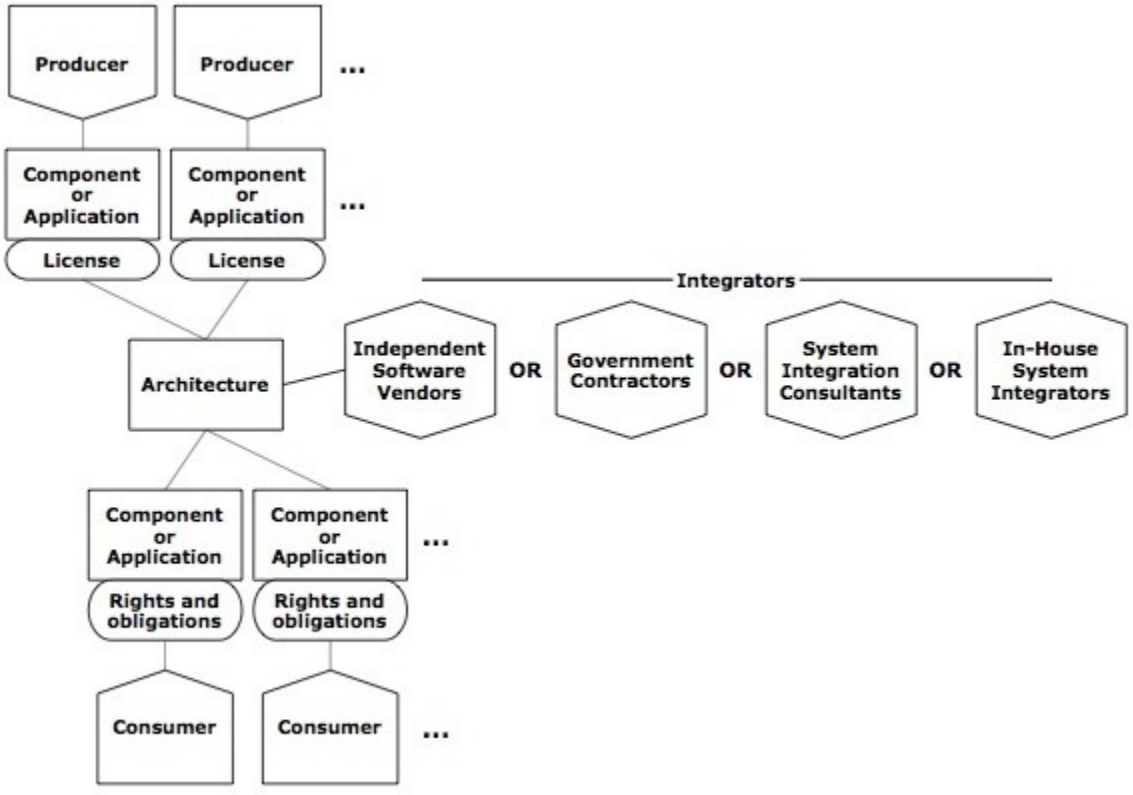


Figure 2: A schematic description of software supply networks for FOSS ecosystems [Alspaugh, Asuncion, *et al.* 2009b].

Networked computer game ecosystems

Participants in this social world focus on the development and evolution of first person shooter (FPS) games (e.g., *Quake Arena, Unreal Tournament*), massive multiplayer online role-playing games (e.g., *World of Warcraft, Lineage, EveOnline, City of Heroes*), and others (e.g., *The Sims* (Electronic Arts), *Grand Theft Auto* (Rockstar Games)). Interest in networked computer games and gaming environments, as well as their single-user counterparts, have exploded in recent years as a major mode of entertainment, popular culture, and global computerization movement. The release of *DOOM*, an early first-person action game, onto the Web as FOSS with a GPL license in 1999, was the landmark event that launched the development and redistribution of computer game *mods* — open source and distributable modifications of commercially available games created with software development kits provided with the retail game package by the game's software developer [Scacchi 2010b]. The end-user license agreement for games that allow for end-user created game mods often stipulate that the core game engine (or retail game software product) is protected as closed source, proprietary software that cannot be examined or redistributed. Such licenses further state that any user-created mod can only be redistributed as open source software that cannot be declared proprietary or sold outright, and must only be distributed in a manner where the retail game product must be owned by any end-user of a game mod. This has created a secondary market for retail game purchases by end-users primarily interested in accessing, studying, playing, further modifying, and redistributing game mods [Scacchi 2004, 2010b].

Mods are variants of proprietary (closed source) computer game engines that provide extension mechanisms like (domain-specific) game scripting languages (e.g., *UnrealScript* for mod development with *Unreal* game engines from Epic Games Inc.) that can be used to modify and extend a game. Extensions to a game created with these mechanisms are published for sharing across the Web with other game players, and are licensed for such distribution in an open source manner. Mods are created by small numbers of users who want and are able to modify games (they possess some software development skills), compared to the huge numbers of players who enthusiastically use the games as provided. The scope of mods has expanded to now 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). For additional background on computer game mods, see http://en.wikipedia.org/wiki/Mod_(computer_gaming).

Perhaps the most widely known and successful game mod is *Counter-Strike* (CS), which is a total conversion of Valve Software's *Half-Life* computer game. Valve Software has since commercialized *CS* and many follow-on versions. *CS* was created by two game modders who were reasonably accomplished students of software development.

Millions of copies of *CS* have subsequently been distributed, and millions of people have played *CS* over the Internet, according to http://counterstrikesource.net/. Other popular computer games that are frequent targets for modding include those based on the *Quake, Unreal, Half-Life,* and *Crysis* game engines, *NeverWinter Nights* for role-playing games, motor racing simulation games (e.g., *GTR* series), and even the massively popular *World of Warcraft* (which only allows for development and sharing of end-user interface mods, but not the game itself). Thousands of game mods are distributed through game mod portals like http://www.MODDB.com. In contrast, large successful game software companies like Electronic Arts and Microsoft do not encourage game modding, and do not provide end-user license agreements that allow game modding, redistribution, or integration with FOSS systems.

Scientific computing ecosystems for 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. In contrast to development-oriented FOSSD ecosystems, open source systems play a significant role in scientific research communities. For example, when scientific findings or discoveries resulting from remotely sensed observations are reported, members of the relevant scientific community want to be assured that the results are not the byproduct of some questionable software calculation or opaque processing trick. In scientific fields like astrophysics that critically depend on software, FOSS is considered an essential precondition for research to proceed, and for scientific findings to be trusted and open to independent review and validation. As discoveries in the physics of deep space are made, this in turn often leads to modification, extension, and new versions of the astronomical software in use. This enables astrophysical scientists to further explore and analyze newly observed phenomena, or to modify/add capabilities to how the remote sensing or astrophysical computation mechanisms operate. For example, the NEMO Stellar Dynamics Package at http://bima.astro.umd.edu/nemo/ is now at version 3.3.0, as of May 2010.

To help understand these matters, consider the deep space image example found at http://antwrp.gsfc.nasa.gov/apod/ap010725.html. This Website page displays a composite image constructed from telemetry data from both X-ray (Chandra Observatory) and optical (Hubble Space Telescope) sensors. The FOSS system processing pipelines for each sensor are mostly distinct and are maintained by different

organizations [Scacchi 2002]. 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 FOSS systems that must be brought into alignment for such an image to be produced, and for such a scientific discovery to be claimed and substantiated.

World Wide Web ecosystems

The SourceForge web portal (http://www.sourceforge.net), the largest associated with the FLOSS community, currently manages information on more than 2M registered users and developers, along with nearly 240K FOSSD projects (as of July 2010). More than 10% of those projects indicate the availability of a mature, released, and actively supported software system. However, some of the most popular FOSS projects have their own family of related projects, grouped within their own ecosystem, such as for the Apache Foundation and Mozilla Foundation. Participants in the ecosystems focus on the development and evolution of systems like the *Apache* web server, *Mozilla/Firefox* Web browser, *GNOME* and *K Development Environment* (KDE) end-user interfaces, the *Eclipse* and *NetBeans* interactive development environments for Java-based Web applications, and thousands of other applications or utilities. This world is the one most typically considered in popular accounts of FOSS projects. The two main software systems that enabled the World Wide Web, the *NCSA Mosaic* Web browser (and its descendants, like *Netscape Navigator*, *Mozilla*, *Firefox*, and variants like *K-Meleon*, *Konqueror*, *SeaMonkey*, and others), and the Apache Web server (originally know as *httpd*) were originally and still remain active FOSSD projects.

The GNU/Linux operating system environment is situated within one of the largest, most complex, and most diverse FOSS ecosystems within this arena, so much so that it merits separate treatment and examination. Many other Internet or Web software 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 directed toward *system administrators* or *software developers* as the targeted user base, rather than the eventual end-users of the resulting 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.

Research findings

The study of software ecosystems is emerging as an exciting new area of systematic investigation and conceptual development within software engineering. Since the

concept first appeared [Messerschmitt and Syzperski 2003], increasing attention has been paid to the many possible roles software ecosystems can play in shaping software engineering. For example, Bosch [2009] builds a conceptual lineage from software product line (SPL) concepts and practices [Bosch 2000; Clements and Northrop 2001] to software ecosystems. SPLs focus on the centralized development of families of related systems from reusable components hosted on a common platform within an intra-organizational base, with the resulting systems either intended for in-house use or commercial deployments. Software ecosystems then are seen to extend this practice to systems hosted on an inter-organizational base, which may resemble development approaches conceived for virtual enterprises for software development [Noll and Scacchi 2001]. Producers of commercial software applications or packages thus need to adapt their development strategy and business model to one focused on coordinating and guiding decentralized software development of their products and enhancements (e.g., plug-in components).

Along with other colleagues [Bosch and Bosch-Sitjsema 2009; Brown and Booch 2002; van Gurp, Prehofer, *et al.* 2010], Bosch identifies alternative ways to connect reusable software components through integration and tight coupling found in SPLs, or via loose coupling using glue code, scripting or other late binding composition schemes found in ecosystems or other decentralized enterprises [Noll and Scacchi 1999, 2001]. This is key to enabling software producers to build systems from diverse sources.

Jansen and colleagues [Jansen, Beinkkemper, *et al.* 2009; Jansen, Finkelstein, *et al.* 2009] observe that software ecosystems (a) embed software supply networks that span multiple organizations, and (b) are embedded within a network of intersecting or overlapping software ecosystems that span the world of software engineering practice. Scacchi [2007] notes that the world of FOSS development is a software ecosystem different from those of commercial software producers, and its supply networks are articulated within a network of FOSS development projects. Networks of FOSS ecosystems have also begun to appear around very large FOSS projects for Web browsers, Web servers, word processors, and others, as well as related interactive development environments like *NetBeans* and *Eclipse*, and these networks have become part of global information infrastructures [Jensen and Scacchi 2005].

We can classify findings in this line of work as research that has investigated the effects of the broader ecosystem on FOSS projects and vice versa.

Effects of the broader ecosystem on FOSS projects

Source of new developers. Ghosh [2006] found that Europe is the leading region of globally active FOSS software developers and global project leaders, followed closely

by North America. Asia and Latin America face disadvantages at least partly due to language barriers, but may have an increasing share of developers active in local communities.

A few studies have gone beyond reports of motives to examine how intrinsic, extrinsic and other factors interact to influence individuals' participation in particular projects. [e.g., David and Shapiro 2008; Roberts, Hann, *et al.* 2006]. For example, by studying 135 projects on SourceForge, Xu, Jones, *et al.* [2009] found that individuals' involvement in FOSS projects depends on both intrinsic motivations (i.e., personal needs, reputation enhancement, skill gaining benefits and fun in coding) and project community factors (i.e., leadership effectiveness, interpersonal relationships and community ideology).

Corporate involvement. Research on this topic has examined the reasons that companies are investing internal resources in FOSS development. For example, Bonaccorsi and Rossi [2006] found that firms are motivated to be involved with FOSS because: 1) it allows smaller firms to innovate, 2) "many eyes" assist them in software development and quality assurance, and 3) to aid in the ideological fight for free software, though this factor comes in at the bottom of the list. In comparison with individuals, they found that firms focus less on social motivations such as reputation and learning benefits.

FOSS ecosystems also exhibit strong relationships between the ongoing evolution of FOSS systems and their developer/user communities, such that the success of one co-depends on the success of the other [Scacchi 2007]. Ven and Mannaert discuss the challenges independent software vendors face in combining FOSS and proprietary components, with emphasis on how FOSS components evolve and are maintained in this context [Ven and Mannaert 2008].

Similarly, by studying the firm-developed innovations embedded within Linux, Henkel [2006] emphasized the importance of receiving outside technical support as a motivator for revealing code. By studying four firms involved with FOSS, Dahlander and Magnusson [2008] discovered three approaches that firms used to connect with FOSS communities: 1) accessing development in the community in order to extend their resource base; 2) aligning their strategy with the work in the community; and 3) assimilating work from the community. Feller, Finnegan, *et al.* [2008] discuss a new form of OSS network involving firms, which use not only IT infrastructure, but also social network.

Licenses. A few empirical studies have taken this framework to examine the influence of license choices on various aspects of FOSS development [German and Hassan 2009; Sen Subramanian, *et al.* 2008; Stewart, Ammeter, *et al.* 2006]. By examining the SourceForge projects, Lerner and Tirole [2005b] examined the relationships between

project types and license choices. For example, they found that highly restrictive licenses are more common for projects geared towards end-users, and significantly less common for projects aimed at software developers.

Tools. Surprisingly little research has examined the use of different software development tools and their impact on FOSS team activities. One exception is Scacchi [2004], who discussed the importance of FOSS-based software version control systems such as *CVS* or *Subversion*, for coordinating development and for mediating control over source code development when multiple developers may be working on any given portion of the code at once. This paper also discussed the interrelation of *CVS* use and email use (i.e. developers checking code into the *CVS* repository discuss the patches via email). Michlmayr and Hill [2003] illustrated the importance of software bug trackers to coordinate FOSS contributors working on resolving questions about unexpected system behaviors.

Government policies towards FOSS software development. Governments around the globe have considered the issue of providing direct or indirect support to FOSS activities, turning FOSS into a political issue [Rossi 2006]. As of September 2006, nearly 100 governments in over 40 countries had taken legislative action in support of FOSS [Lee 2006; Lewis, 2010]. Countries as far apart as Germany, Brazil, Italy and Singapore, among others, have all endorsed FOSS software to some extent, according a preference for adoption of FOSS in governmental offices, offering temporary tax reductions and financial grants to fund Linux-related projects or through some other means [Hahn 2002 from Rossi 2006]. Chan [2004] examined the practices that surround the emergence of free software legislation in Peru, observing a shift in framing beyond free software's economic and technical merit, asserting the need to overcome the dominance of privileged nations and corporate interests that have infiltrated Peruvian government. Free software was identified as a tool for smaller nations to address their limitations and position within the global market. Further, as noted above, free software offered the potential to both free the state from the clutches of corporate interests and increase public participation in a malleable, reprogrammable political decision-making process.

Cultural impacts. Verma, Jin, *et al.* [2005] explored the factors that influence FOSS adoption and use within two different FOSS communities, one in the U.S. and one in India. They found that the degree of compatibility with users' mode of work, and ease of use are the two significant factors that influence FOSS use in the U.S. FOSS community. For the Indian community, compatibility is the only significant factor. Through their comparison of FOSS developers in North American, China and India, Subramanyam and Xia [2008] found that developers in different regions with similar project preferences are driven by different motivations. For instance, for projects that are larger in scale, more modular and universal in nature, Chinese developers are found

to be drawn by intrinsic motives while Indian developers are found to be mostly motivated by extrinsic motives.

Effects of FOSS on the broader ecosystem

FOSS is a venue for training developers. Consistent with the view of FOSS projects as an opportunity for training developers are the results of surveys showing motives such as career development [Hann, Roberts, *et al.* 2002, 2005; Hars and Ou 2002; Orman 2008] or learning opportunities [Shah 2006; Ye and Kishida 2003] as commonly mentioned motivations for participation in projects. Programs such as Google's *Summer of Code* (see http://code.google.com/soc/) have emerged with this goal in mind.

Many tools are open source, and adoption of new tools has greatly facilitated development. The United Nations Conference on Trade and Development reported [2004] that FOSS provides an environment for the development of local industry and skills, noting that FOSS processes and methodologies have served to influence technology development in general.

License. FOSS has led to innovations in licensing models. For example, the Creative Commons licenses are reported to have been inspired in part by Free Software licenses (see http://creativecommons.org/about/history/). FOSS licenses such as version 3 of the GNU General Public License (GPL3 — see http://www.gnu.org/licenses/gpl.html), further stipulate terms explicitly conveying a grant of patent licensing for software covered under its terms. The Apache License, Version 2 (see http://apache.org/licenses/LICENSE-2.0) includes similar patent terms. While software patents remain a controversial and legally ambiguous subject, this move extended the terms of licensing beyond copyright to other forms of intellectual property. These sorts of protections are of high interest both to producers of FOSS, as well as consumers. Moreover, they naturally raise questions regarding software license compatibility.

Lessig [2006] argued that the code that makes up a system governs how we interact with that system. At present, the Open Source Initiative has approved more than 50 open source licenses. The increasing number of licenses is a phenomenon frequently referred to as license proliferation [Gomulkiewicz 2009]. Recent research has begun to examine license mismatch [German and Hassan 2009] and compatibility and the effect on the ecosystem [Alspaugh, Asuncion, *et al.* 2009b].

FOSS has led to innovative models for coordination and governance of collaborative work. Relying on four in-depth case studies of firms involved with open source software, we investigate how firms make use of open source communities, and how that use is associated with their business models [Dahlander and Magnusson 2008]. This paper

analyzes a dynamic mixed duopoly in which a profit-maximizing competitor interacts with a competitor that prices at zero (or marginal) cost, with the accumulation of output affecting their relative positions over time. The modeling effort is motivated by interactions between *Linux*, a FOSS operating system, and Microsoft's proprietary, closed source operating system *Windows,* which consequently emphasizes demand-side learning effects that generate dynamic scale economies (or network externalities) [Casadesus-Masanell and Ghemawat 2006].

Boucharas and colleagues [2009] draw attention to the need to more systematically and formally model the contours of software supply networks, ecosystems, and networks of ecosystems. Such a formal modeling base may then help in systematically reasoning about what kinds of relationships or strategies may arise within a software ecosystem. For example, Kuehnel [2008] examines how Microsoft's software ecosystem developed around in its *Windows* operating systems and key applications (e.g., *Office*) may be transforming from "predator" to "prey" in its effort to control the expansion of its markets to accommodate OSS (as the extant prey) that eschew closed source software with proprietary software licenses.

Code reuse. Haefliger, von Krogh, *et al.* [2008] found that code reuse is extensive across a sample of code and that FOSS developers, much like developers in firms, apply tools that lower their search costs for knowledge and code, assess the quality of software components, and have incentives to reuse code. There has been an increase in the number of proprietary systems taking in process outputs of FOSS systems as inputs into their own processes [Riehle, Ellenberger, *et al.* 2009]. Another study showed that software organizations can achieve some economic gains in terms of software development productivity and product quality if they implement OSS components reuse adoption in a systematic way [Ajila and Wu 2007].

Open research questions

The research reviewed above suggests that there are rich and complex relations between FOSS projects and a diverse set of actors in the surrounding ecosystem. However, it is clear that this research has provided only a glimpse of the ecosystem web, and further research is needed to provide robust theories that are explanatory, predictive, and transferable. Research questions about FOSS ecosystems seek to elicit new knowledge about the relation between FOSS projects and:

1. Users (individual and organizational)

- In what ways does public participation affect FOSS development projects?

- What factors have led to organizational adoption or non-adoption of FOSS (i.e., adoption of *SVN*, *Linux*, etc. in enterprise) and what are the implications for projects?

2. Developers:

- What are the conditions for active participation in FOSS projects (enskillment process, conditions for engagement and sustained collaboration)?

- How do developers learn to work in FOSS projects? What role does formal education play?

- Why does FOSS apparently reduce the diversity of participation, with lower participation from women and minorities, as compared with other software work? What can be done to facilitate or encourage participation?

- What is the impact of FOSS on international labor markets for programmers and IT workers more generally? Does FOSS undermine or enable the US IT worker?

- How does the fact that reputation is vested in the individual alter the labor market? (Companies pay for much of the labor in many open source projects, but reputation vests in the individuals, who can readily move to other companies, or even to independent projects using the same source they worked on in the company.)

3. Tools:

- How are the practices and philosophy of FOSS instantiated within its collaboration technologies?

- How does the introduction of new tools realize technology-led organizational change?

4. Other projects and their code:

- What elements comprise a software system?

- What are the relationships between these elements?

- How do they interact?

- What are the processes that describe their interaction?

- Can we identify patterns or classes of relationships between elements of the system?

- How do these relationships change over time and why?

- What are the effects of the heterogeneity of commercial, free/open, non-profit, government, and other organizational forms on the interactions of projects?

5. Companies and industries:

- What is the impact of FOSS on the software industry and vice versa?

- Has FOSS enabled the US software industry to move up the value chain and driven down business start-up costs to facilitate a new way to try business ideas?

- Does FOSS destroy business models, de-monetize collective activity, and devalue intellectual work?

6. Institutions:

- What are the effects of institutions (e.g., social norms, formalized rules, governance requirements) that may be enforced through the connection to an overarching nonprofit foundation on collaboration at the project level?

- How is such an ecosystem governed? How do elements of such ecosystems synchronize and stabilize, or desynchronize and destabilize (e.g., to achieve and effect competitive advantage)?

7. Legal regimes:

- How does the influence of various open source licenses construct, or destruct the collaborative environment? E.g., do various license terms restrict how “open” code can be used by other projects thus affecting collaboration?

- How do national copyright and patent policies influence collaboration on open source projects? Do different policies lead to different success rates or possibly different types of FOSS?

8. Government policies (e.g., procurement, patent, copyright, etc.):

- How do the policies of one country influence government policies in another?
- How does government policy promote or inhibit collaboration in communities?
- How do public policies enhance or impede the natural tendency to collaborate within a given community?
- What are the risks and implications of a policy of “open sourcing” all products of government-funded research?

9. Cultures:

- As FOSS moves beyond its US and European cultural roots, will it be a successful model for collaboration? Will FOSS itself learn from the new and diverse cultures it increasingly encounters?
- How do the cultural differences of various developers impact their involvement in a project? On a larger scale, how do various project cultures interact in a collaborative environment?
- Is FOSS fostering a culture of copying? Doing rather than asking permission from gatekeepers?
- How do FOSS-like principles relate to or influence broader democratic principles, or effect new models of public participation?

10. Other open movements:

- Why is FOSS so often cited as an inspiration wherever people work together online? Is its influence only as metaphor?
- How is the visibility and transparency of FOSS an aspect of its influence?
- What has shaped the popular understanding of Open Source? Does it matter if that understanding is erroneous?
- How are FOSS collaborative principles transferring to other collaborative domains outside of software? How similar is FOSS to these other kinds of efforts?

- What lessons from FOSS can inform how we might undertake “open” collaborations in other areas (e.g., collaboration in science)?

- What are the limits of the FOSS way of working? Can it usefully extend beyond its software origins? What innovation might be prompted by the attempt?
- How can work be designed to be suitable for a FOSS-like mode of production (e.g. Wikipedia)?

- Which practices work when undertaking socio-technical redesign? What are the mechanisms by which ideas transfer from FOSS to other kinds of projects (and vice versa)?

Conclusions

The emergence of FOSS is transforming software production. It is transforming the interaction between organizations, advancing science, entertainment, economics, government, and society. Studies to date suggest these transformations are as complex and far reaching as they are difficult to track. At present, we lack the tools to understand the nature of these transformations. Similarly, we lack a deep understanding for how FOSS is transforming mainstream software production within enterprises, the global software/IT industry, government agencies, and society at large.

Given the diversity of factors, studies of FOSS must engage a variety of discipline areas, such as behavioral economics, law, science and technology studies, ethics, communications, anthropology, computing, political science, behavioral sciences, sociology, public planning and policy. These research areas could employ a variety of research methods and would benefit from multidisciplinary approaches to FOSS as a socio-technical system. However, multidisciplinary research is hard to do. It requires researchers with an openness and willingness to engage with others. Furthermore, it requires disciplinary scientists to master the details of the domain.

There is a particular opportunity for FOSS to be an exemplar for new practices for sustaining NSF funded research and transferring results of that research. Simply throwing code on a server might be called “open sourcing”, yet that seems quite insufficient to achieve the hoped-for benefits. For example, without indefinite sources of funds (often longer than the typical horizon of an NSF grant), the data and other products may not be archived sustainably, damaging the accessibility of these results to future researchers. Open source may offer new models for dissemination and sustainability of scientific research.

Evolution

Overview

An evolutionary perspective on FOSS leads to fruitful insights into the science of open source systems.

One of the hallmarks of FOSS systems is their strongly fluid and dynamic character . All types of open source systems — from wikis, blogs, and open media, to network organizations, ad hoc teams of volunteers, and community groups — manifest adaptive, continuously changing, and surprising emergent behaviors. The fluid character of these systems might be partly due to the networked environment in which they operate, as both the hardware and software environment of digital artifacts are in “constant flux” [Allison, Currall, *et al*. 2005, p. 368]. More importantly, however, it is the *drivers* of change that differentiate them from traditional systems. The open, collective, and participatory aspects of these systems seem to drive their dynamics, while the electronic environment of their implementation supports and sustains them. FOSS demonstrates this state of constant flux quite vividly: changes in the embedding environment and context of use lead to changes in requirements; design features and functions evolve; hardware systems, platforms, and properties are redesigned; team members move, relocate, or refocus; legal arrangements and license agreements get revised; and so on [Ekbia and Gasser 2008]. Change, in short, is the rule rather than the exception. Our current challenge is that we do not fully understand the interactions and high-order effects of this change. There is no common temporal frame of reference to synchronize the evolution of these systems and subsystems. Rather, each system runs on its own “clock,” changing at different rates and with different bindings to external schedules and clocks [Ekbia 2009]. This characteristic gives rise to an interesting similarity between open source systems and biological systems.

Evolutionary biology distinguishes “development” from “evolution” — the former covers the ongoing adaptations that arise during one life cycle, while the latter covers transformation or mutation propagated across generations. This is a useful distinction with regard to FOSS systems, where development refers to within-release revisions and adjustments, while evolution characterizes what happens across major releases. The significance of this distinction becomes evident once we notice, for instance, the common software development methodologies described by Boehm [2006] earlier in this report.

When studying software evolution, it is necessary to clarify whether attention is directed at development of a given system throughout its life, or at the evolution of software technologies over generations that are disseminated across multiple populations. It appears that many of the studies concerned said to be concerned with “software evolution” are in fact studies of patterns of development of specific systems, rather than patterns of evolution across different systems, particularly as compared to work in biological evolution. However, the laws and theory of software evolution articulated by Lehman and associates [Lehman 1980; Lehman and Belady 1985] 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.

Considering the subject of FOSS system evolution at a macro level, it appears that there are no widely cited studies that examine the issues of memes, competition, resource scarcity, population density, legitimacy, and organizational ecology that characterize evolution as a social and cultural process [Christiansen and Kirby 2003; Gabora 1997; Hannan and Carroll 1992; Saviotti and Mani 1995]. The study of software evolution in general, and FOSS evolution in particular, is still in its infancy. Similarly, sophisticated mechanisms for cross-project (or lateral) innovation and resistive entrenchment, as seen in recent evolutionary models [Gould 2002; Sapp 2009; Wimsatt and Schank 2004], are not yet articulated for FOSS systems. Studies and insights from these arenas are yet to appear, and thus need to be explored.

Conventional closed source software systems developed within centralized corporate productive units, and FOSS systems developed within globally decentralized settings without a corporate locale represent alternative technological regimes. Each represents a different technical system of production, distribution or retailing, consumption, differentiated product lines, dominant product designs and more [Hughes 1987]. Similarly, software development methods based on object-oriented design and coding, agile development, and extreme programming entail some form of alternative technological regime [Nelson and Winter 1985; Scacchi 2006]. Concepts from theories of technological evolution, and observations about 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 production regimes, including the socio-technical regime of FOSS system development and evolution.

What is missing?

An essential requirement for this standard model is that the successive representations made through the software life cycle — the progressive transformations from user

needs to reified requirements, from requirements through interpretations of designs and code, from written codes to compiled, linked, assembled and delivered products, and from packaged products through products-in-use reflecting actual specific behaviors — are “meaning preserving.” In this view, the late-phase delivered products must accurately reflect the early-stage goals, assumptions, and specifications. Many of the cost, time, and quality liabilities of modern software production spring from the need to sustain the accuracy of these transformations. Many development tools and practices explicitly aim at increasing the reliability of this correspondence in the face of very complex, distributed development processes with many stakeholders and contingencies both internal and external. In short, software, in this conception, is a reified object with a life cycle that goes through phases of birth, growth, maturation, and death. The eventual destiny of the software is by and large in the hands of designers (upstream) and, to some extent, business people and the market (downstream).The survival of the best and the fittest software seems to be guaranteed in a kind of Darwinian process. However, this Darwinian model provides an idealized, naive image of software development processes, missing important aspects unaccounted for, and leaving important questions unanswered. The parallel with biology is again useful here. The Darwinian theory of natural selection explains the evolution of large organisms, but it fails to explain the evolution of a much larger subset of the natural world — namely, microbes — the evolution of which spanned the first 3.5 billion years of the overall 4.5 billion years of the emergence of life on earth. This huge evolutionary machinery is still at work, inciting biologists to formulate non-Darwinian theories of evolution that would fill in the theoretical gap [Sapp 2009]. Similarly, recent replication of Darwin's studies of Finches on the Galapagos finds that natural selection may in fact be a rapid adaptation mechanism observable in human-scale timeframes [Weiner 1995], rather than the slow progressive process that Darwin posited. So again, our understanding of evolutionary processes is being continuous improved and refined. But a reliance on the traditional Darwinian model increasingly limits our understanding of the new evidence and studies now at hand. By way of analogy, our limited understanding of FOSS evolution does not allow us to explain or predict:

- How do the changes in FOSS structure, code, hardware, platform, community, legal environment, and so on influence, constrain, or amplify each other, across major and minor system releases?

- What have we learned about how to develop long-lived FOSS systems — systems that survive transitions across generations of releases and platforms?

- Under what conditions or circumstances is FOSS code the best way to capture and convey knowledge about how a program evolves? Are there other mechanisms that might be appropriate?

- How does the structure or architecture of long-lived FOSS systems change across versions, releases, or processing platforms? Some of these FOSS systems were originally developed on computers like DEC VAX 11/780 that probably no longer exist, and whose processor performance might have been measured in the range of 1M instructions per second (1Mips) or more, compared to the multi-gigahertz processors of today, and the multi-core, massively parallel processors of tomorrow.

- How does computer processor performance affect the evolution of long-lived FOSS systems?

- How do changes in processor architecture (from single core, to multi-core, to massively parallel) mediate the evolution of long-lived FOSS systems?

- Under what conditions is it necessary to create emulators of vintage computer systems in order to revive or sustain legacy source code? Consider for example the emergence of the FOSS-based *MAME* (multi-arcade machine emulator) system that emulates vintage Motorola 6502 compatible game arcade computers that in turn allows thousands of vintage arcade games to be revitalized and played on modern PCs [Scacchi 2004].

- Do software systems in general tend to be designed assuming a particular processor architecture, such that using them on newer architectures may result in large parts of the source no longer usable or relevant? If X-Windows was developed to support graphic stroke (Tektronix 4014) terminals, or computers that lack graphics acceleration cards/processors now commonly available today, what does this mean for the evolution of its source code?

- When does a software product line come to an end? When does a long-lived FOSS system evolve into a hobby for nostalgic software developers?

- What roles do online artifacts take in sustaining the ongoing development of FOSS systems? Such artifacts are, for example, FOSS project discussion forums, group blogs, CVS/Subversion logs, IRC transcripts, and project digests (“Kernel Cousins”). They are often knowledge-rich and highly contextualized. They stand in contrast to the recommended practice of software engineering that stresses the creation of formal representations of software, like “requirements specifications” and “language-based design notations” [Sommerville 2006].

In short, the general problem area is: how and why do FOSSD outcomes, activities, technologies, infrastructures, etc. develop and change over time; do these changes follow specific patterns or principles; and what evolutionary trajectories are typical with

FOSS as compared to other forms of software? Analyzing such evolutionary diversification is significant in multiple ways. It can offer new ways to explain and predict likely outcomes for certain initiatives. It helps us understand how software processes learn and change, what events and factors lead to such change, and how much change is driven by social, regulatory, technical or market changes. These are valuable in managing and coordinating FOSS production efforts, formulating policy and regulatory interventions, and understanding the pattern of software change in relation to its environment. Evolutionary analyses offer insight into how software and its related processes,infrastructures, and tools co-evolve over time . In addition, these analyses help discern factors that lead either to the growth or decline in FOSS initiatives, and what factors influence the overall change patterns in FOSS communities.

What do we currently know?

The subject of software evolution is one of the longest standing topics for empirical software engineering, with the earliest studies appearing in the late 1970's, while ongoing studies continue to this day [Godfrey and German 2008; Lehman 1980; Lehman and Belady 1985; Madhavji, Ramil, *et al.* 2006]. Studies of FOSS evolution have appeared since about 2000, and surveys of this work have already begun to appear [Robles, Gonzalez-Bahrona, *et al*. 2003, Koch 2005, Ye, Nakajoki, *et al.* 2005, Scacchi 2006, Deshpande and Riehle 2008, Fernandez-Ramil, Izquierdo-Cortazar, *et al.* 2009]. Noteworthy about these studies of FOSS evolution are (a) the discovery of large numbers of FOSS systems that are growing at sustained superlinear (exponential) rates, and (b) that FOSS code and developer/user communities co-evolve together, rather than independently. However, such growth is not inevitable, nor it is insured, as other FOSS systems do not show such sustained evolutionary growth patterns.

A number of the current studies of FOSS evolution focus on software products such as source code releases, application systems or families, development tools and infrastructure, and process models. Other studies examining the evolution of software development artifacts, practices, and project communities have appeared in smaller numbers compared to those focusing on the software itself. Studies of FOSS ecosystems are just beginning to appear [Alspaugh, Asuncion, *et al.* 2009b]. New methods for studying FOSS evolution have appeared, including methods relying on data mining tools and techniques applied to FOSS project repositories, whether for specific large projects, or those analyzing FOSS evolutionary patterns found in multi-project FOSS repositories like *SourceForge*, or archives of such repository data found at *FLOSSmole* [Howison, Conklin, *et al*. 2006], the *SourceForge Data Repository* at Notre Dame University, and elsewhere [Gasser and Scacchi 2008]. As a result, we can now see different research strategies and methods coming into view which in turn reflect the different kinds of studies of FOSS evolution that can now be undertaken. These include:

- Very large, population-scale studies examining common objects selected and extracted from hundreds to tens-of-thousands of F/OSS projects, or surveys of comparable numbers of F/OSS developers.

- Large-scale cross-analyses of project and artifact characteristics, such as code size and code change evolution, development group size, composition and organization, or development processes.

- Medium-scale comparative studies across multiple kinds of F/OSS projects within different communities, use or production ecology, or software system types.

- Smaller-scale in-depth empirical studies of specific F/OSS practices and processes for ethnographic study or hypothesis development, as well as the in-depth investigation of details of different socio-technical resources and relationships.

Details identifying studies at each scaling level of data sampling and analysis can be found in the survey by Gasser and Scacchi [2008]. Studies of software evolution prior to 2000 seem to be limited to either medium or small-scale studies, as large-scale and very large-scale data sets were previously not available. Studies of FOSS system evolution at different scales thus represent a new opportunity for discovering scientific knowledge about software evolution through new data sources.

Next, there are several important distinctions to be made about evolution of FOSS systems. First, analogous to models in evolutionary biology, one level is involved with analyzing patterns of developmental change for singular processes, artifacts, projects or communities within a generational scope. This kind of evolutionary adaptation is sometimes called software maintenance or continuous development. Second, we can seek to observe evolutionary patterns across processes, artifacts, projects or communities in how they build an ecosystem, adapt to specific ecological niches or maintain specific environmental interactions as some sort of “species” or stage of evolution across generations. Evolution in this regard covers generational changes that are expressed across software releases, platforms, contributors, and the like. Subsequently, the study of evolution of FOSS systems needs to be able to distinguish these two levels of change and also understand their interactions.

Last, a FOSS system can be viewed as an evolving entity which involves many interacting and evolving components. Such components involve participants and their patterns of interaction (size, volatility, intensity, etc.), structure of the community in terms of organization and organizing principles, decision rights, principles of ownership, value extraction and property rights (licensing), the nature of artifacts produced (artifacts),

technological artifacts and capabilities appropriated or prescribed as part of the software production work (software development tools, software version/configuration management tools), software production processes and related methods, the nature of the infrastructures in which the FOSS participates, the structure and change of the communities, and changes in the broader environments in which the FOSS projects operate.

What do we need to know?

We now turn to identifying where future research studies of FOSS evolution will appear, and the kinds of deep, challenging, and fundamental problems they can address. These studies will address FOSS evolution at the level of: products and artifacts; software property licenses; development practices, processes, and tools; development infrastructures, project communities; and socio-technical environment or ecosystem.

Are these informal FOSS development artifacts merely containers of information, or are they, along with the tools that support their creation and management (like project wikis or discussion forums) also the workplace where FOSS development work happens? If so, the evolution of FOSS development work practices might be understood through examination of the evolution of the patterns of usage for creating and sustaining these artifacts.

If online artifacts are the new workplace for globally dispersed and decentralized FOSS development, then how will the evolution of these objects of interaction shape the software development workplace of the future? How will the future of software development work determine which artifacts will be most critical to the ongoing development of a FOSS project? How is the development of FOSS best characterized from a software process perspective? How is the process of software evolution, perhaps the oldest, empirically studied problem in the field of software engineering, changing to account for FOSS? Empirical studies of software evolution prior to 2000 were almost exclusively focused on commercial software systems developed in proprietary settings that were deprecated over time, and subject to extant software business models and market conditions. Since 2000, nearly all studies of software evolution practices, processes, and artifacts focus on the empirical study of FOSSD projects, mainly due to the public accessibility of FOSS development and evolution data [Scacchi 2006].

Thus, the problem of understanding the evolution of FOSS system code and artifacts will be situated and understood with reference to development processes, and project forms. Such investigation remains an open problem area that is highly amenable to empirical study, and it is a problem area that is likely to be long-lasting in its importance

to the future of software and the future of closely aligned social practices that embrace an open source-like approach (e.g., open science, open content creation).

Can we create general laws of software evolution primarily from studying FOSS? Studies of proprietary software have usually been subjected to confidentiality restrictions such as Non-Disclosure Agreements that limit the details that can be published. This poses two fundamental problems: first, many empirical studies of software evolution cannot easily be replicated or extended due to lack of open access to the data collected in proprietary settings; and second, so much detail is typically left out of the study that we are asked to take a lot on faith. These studies ask us to trust that the researchers have done the study properly, and also to accept the results without fully understanding the design or even domain of the target system. Some FOSS have been around for more than a decade or two, and have preserved their histories. The availability of such FOSS systems has thus initiated a golden age of research in software evolution, where any researcher can now have access to rich histories of software systems to study. Furthermore, empirical studies of FOSS systems can now be replicated to either confirm or revise previous results, or present discoveries that might have been previously overlooked.

We do know that the goals, processes, economics, and even politics of FOSSD are strikingly different from that of most proprietary systems. What we must ask is: How do the differences affect the resulting systems, and are there significant differences with results from non-FOSS system development efforts [Paulson, Succi, *et al*. 2004]? Finally, we note that just as there are a wide variety of industrial software development processes, there are also many development models that fit under the umbrella of "open source": some projects are driven mainly by part-time enthusiasts, and decisions are made by an informal consensus; some are initiated by companies but then donated (or abandoned) to the broader user community; and some are supported by organizations that spearhead, oversee, and legally represent official development efforts through for-profit corporations or nonprofit foundations. This last category — which includes such well known systems as *Mozilla*, *MySQL*, *Eclipse*, and *Mono* — is particularly interesting as the organization usually hires and manages most of the key developers and so functions within a kind of hybrid process model: the system is designed, developed, and managed by a formal organization, but development is "out in the open" [cf. Dinkelacker, Garg, *et al*. 2002]. Perhaps these kinds of systems hold the key to an improved understanding of the differences and similarities between FOSS and proprietary closed source software.

While the recent explosion in the amount of empirical work on FOSS systems is good news to the research field, the question must be asked: Is the evolution of FOSS a good indicator of the evolution of software in general? If not, what are the differences? To answer this question, we categorize the evolution of FOSS into five key areas:

processes, practices, and project forms; infrastructure; community; ecosystem; and licensing.

Evolution of FOSSD Processes, Practices and Project Forms

FOSSD processes and work practices evolve as development tools change, the community discovers new ways of coordination and organization, and the rules of governance are modified. Recurring FOSSD work practices represent instances of FOSSD processes within a given FOSSD project organization. So adaptive or responsive changes in recurring, collaborative FOSSD work practices are partly characteristic for how FOSSD processes evolve, and how they jointly co-evolve with the FOSS system and project organization at hand. Outstanding research questions associated with FOSSD processes, work practices and tools include:

- How do we best to discover, recover, and model FOSSD processes that are decentralized and globally dispersed, and that yield useful software systems?

- How do FOSSD processes or practices give rise to sustained evolutionary growth of source code and developer/user communities? Why do some FOSS systems undergo sustained growth at superlinear rates, while others do not?

- How do development processes evolve over time, and what factors affect their evolution? Do development processes change with emerging changes in work task, tools, participants, and as project scale increases? What is the role of alternative project organizational forms in facilitating or inhibiting such changes?

- Do development processes follow incremental or punctuated change patterns? Under what conditions are the changes incremental or punctuated?

- How do tools change and evolve as development processes, work practices, or project forms evolve? How are new tools integrated into processes, practices, or projects, and with what consequences?

- What aspects or elements of FOSSD are critical to analyze in the FOSSD project evolution including passages across new actors, tools, organizational forms, governance regimes, size, task volatility, etc?

The evolution of FOSSD processes and practices encompasses the frequency and distribution of events such as updates to platforms, tools and requirements, and participants or user populations engaging in development processes. The sequencing of

activities in which these components or actors participate (i.e. order of sequences, proportion of different types of activities) is also of interest. Traces of work practices can also be used for analyzing similarities and differences across a set of processes within FOSSD projects, and compared to proprietary software development. One can analyze developmental change in a single process or set of development processes over time in terms of changes in their organization and its variance. There are also options to compare types of development processes and their stages of evolution. Similarly, we can analyze integration and expansion of tool support to address change in production, collaboration or negotiation tasks, and how they influence the process organization,structure, and outcomes.

This requires developing categories to describe and classify process data and events and their connections in a systematic way. It may demand creation of new tool and data collection capabilities to capture in real time process events and their sequences, and to visualize them for analysis. Additional approaches that can use event data may involve simulation, or with larger data sets use of variance based statistical techniques (regression, non-linear regression) to analyze interactions between process features and outcomes like software size, quality, user growth, and the like. Consequently, puzzling questions arise around issues such as:

- How do developer and contributor practices change as a function of the development state of the project (concept, pre-alpha, alpha, beta, or stable releases)? Do these mirror changes in proprietary development practices?

- Do programming language features fragment during the project life cycle [Krein, MacLean, *et al*. 2009]? In other words, is there a general trend towards high or low language entropy throughout the life of a project? Is this good? Can we quantify the benefit of a project moving in either direction?

- Is there a critical mass of communication or socio-technical interaction required to move a project into a phase of rapid, sustainable and exponential growth? If so, which kinds of communication (email, chat, mailing lists, blogs, wikis, phone, face-to-face) or have the most power to support this kind of growth?

- What are the key communication practices, processes, software tools/applications, online artifacts, project forms, or other resources that enable a decentralized, international FOSSD project? Can we see a difference in successful and unsuccessful projects along these lines? How soon in the life cycle of the project do these communication habits need to manifest in order to allow a project to grow at a super-linear rate? Is there a drop-dead point after which the project will collapse under its own weight if not supported by these resources?

A beneficial step would be to develop a collaborative resource to curate and house research data in a single repository like *FLOSSmole*. Once evolutionary data from multiple sources are co-located, it becomes much easier to analyze evolution as a web of processes and practices that are situated within a FOSSD project and its larger surrounding or embedding ecosystem.

Evolution of FOSSD Project Infrastructure

Many products of FOSSD projects have become essential, critical cyberinfrastructure, such as Web servers, Web browsers, data management repositories, interactive development environments, and more. Familiar examples include *Bind*, the *Apache* web server, and the *Eclipse* IDE (where greater than 70% of all current Java development is done using Eclipse). The infrastructure management issues raised above clearly apply to FOSS products. To understand infrastructure evolution, we need conceptualizations of infrastructure's critical dimensions. A model presented by Star and Ruhleder [1996] includes the following dimensions, adapted here to FOSSD projects:

- *Embeddedness*, as indicated by the size, diversity, and complexity of resources and social arrangements that situate a FOSS system.

- *Transparency*, as indicated by the level and extent of openness, as well as the ability to study, modify, and share knowledge or practices for how a given FOSS system is being developed or used.

- *Learned as part of membership*, since participation in and contribution to a FOSSD project demands effort to acquire and make sense of disparate information and online artifacts, to figure out how internal system computations and external development activities are performed.

- *Linked with conventions of practice,* such as how and where to ask questions that elicit knowledge from others, how to collaborate with other FOSSD project contributors, or how to focus on resolving software bugs rather than chiding those who may have contributed them.

- *Embodiment of standards,* particularly in the form of preferred programming/scripting languages, development tools, project repositories, online artifacts, data representation formats, or data communication protocols to use when contributing to a FOSSD project.

- *Built on an installed base,* including participant contributed hardware, software, and networking resources, along with shared FOSSD project repositories.

- *Becomes visible upon breakdown*, since the hardware, software, and project platforms, as well as the beliefs, values, and norms that are shared by FOSSD project participants, help shape when FOSSD practices, processes, or project forms are working well or becoming problematic and conflict-laden.

Star and Ruhleder [1996] present these as characteristics of mature information infrastructures. But such infrastructures have life cycles of creation, growth, maturation, sustenance, and eventual dissolution. It is critical to understand the roles of different FOSS resources in the life cycles of infrastructures along each of the dimensions above. For example, how do dense networks of dependencies between FOSS systems and project resources develop and dissolve? What are the roles of FOSS practices, processes, and project forms in establishing and evolving local or global infrastructural standards? How do FOSS practices, processes, and project forms become invisible and then return to visibility? Similarly, closely related research questions arise when FOSSD projects are expected to create cyber-infrastructure for mission-critical applications in science, industry, or government settings:

- *Creation*: how do new infrastructures come about?

- *Sustainability*: how can infrastructures best be maintained?

- *Vulnerability*: how can infrastructures be attacked or fail, and how can they be protected against such events?

- *Innovation*: how can infrastructures be revised, updated, improved, or decommissioned?

We lack the requisite scientific knowledge to explain, predict, and control how FOSSD project infrastructures are developed and evolved over time and across multiple projects.

Evolution of FOSSD Project Communities

Communities begin, grow, adapt, maintain themselves, and decay within a socio-historical environment and ecology of other communities and use. Some communities compete for members, while others find synergies or cooperative strategies. The social structures of these communities are important to understand, as is their dynamic path

over time. These paths include the interaction patterns of members within and across FOSSD projects.

It would be useful to instrument existing FOSS communities to obtain rich enough datasets to understand these internal and external interactions. To achieve this goal will require (a) a research infrastructure project to instrument FOSS activity and to archive such datasets. Once we have these datasets of multiple community efforts, we can also (b) provide visualizations of this activity, (c) understand the activities and interactions in time, and (d) provide additional tools to augment these activities. Important research questions here include:

- What are the relationships between FOSS system structural characteristics, community interaction, productivity, and use? Do these relationships change over time as the project grows, adapts to its environment, attempts to maintain itself, and decays?

- Where are practice innovations created and developed in OSS projects and OSS ecologies, in terms of structural and dynamic characteristics?

- What information flows best enable productive communication in OSS projects?

- How do FOSS code and project communities co-evolve?

- What augmentation mechanisms, such as visualizations and information digests, provide information and activity summarization and understanding? What are their advantages and problems?

- What tools can augment communication among stakeholders? How might they be designed appropriately, and what are their issues in use?

- Can social or technical mechanisms be designed to identify emerging innovative practices, and best practices developing within a project? What mechanisms could help spread the inter-project communication of those practices?

FOSS communities are often more diffuse, geographically distributed, and artifact-based than other communities, even those online. They exist in clear ecologies of communities, where the interactions and communication flows among them may be critical to their effectiveness and persistence.

Evolution of FOSS Ecosystems

Software ecosystems and their evolution have become a prominent feature of FOSS projects and projects integrating FOSS and non-FOSS licensed components. Such ecosystems connect software producers and software consumers through system architectures created by system integrators. Researchers have found that traditional homogeneously-licensed FOSS projects involve a complex network of producers, who occasionally introduce an unexpected license into the system [Gonzalez-Barahona, Robles, *et al*. 2009]. A software ecosystem evolves as the configuration of a system changes to incorporate different libraries and components from other producers. In some cases, producer projects (such as *Linux Kerne*l and *GNOME*) coordinate their evolution and releases [Alspaugh, Asuncion, *et al*. 2009a]. The health of a FOSS system depends not only on the project that produces or integrates it, but on the web of producers it depends on and the consumers who use it [Alspaugh, Asuncion, *et al*. 2009b; Godfrey and German 2008]. The management of these ecosystems is an important consideration in the predictable evolution of the systems involved.

As innovative technologies evolve and mature they attract other applications that build on them and these interdependencies create an ecology that takes on characteristics of an infrastructure technology. Thus interesting research questions related to this phenomenon include:

- What environmental catalysts allow a group of FOSSD projects to coalesce into a critical mass?
- How does the interaction between users and technology propel some FOSSD projects forward?
- What are significant characteristics of the ecosystems of FOSS systems that evolve at superlinear rates?
- What tools encourage intermittent participation and facilitate learning and collaboration for populations of transient and diverse participants?
- Is there a recurring cycle of growth and decay that drives the success of FOSSD projects? Is this cycle impacted by competition, demand, new legislation, economic policies, IP, standards, etc?
- Will a historical analysis of other seminal projects uncover critical inflexion points at which projects take off? Are there certain governance norms and structures that are best of breed?

In recent years, an increasing number of businesses are shifting some development efforts to FOSSD projects in order to reap greater economic efficiency. Any organization that seeks non-differentiating software can benefit from shifting some development to FOSS communities [Perens 2005]. However, not much effort has been directed towards understanding new socio-technical practices that inevitably emerge due to differences between the existing OSS communities and the organizations that join them [Lin 2005]. Emerging research questions include:

- What factors drive such alliances, and how do these alliances form? Are there common structures for such alliances?
- What factors improve or damage the coordination, collaboration and governance processes in these communities of practice?
- How will these alliances impact the future of the software industry?

Methodologically, we need more grounded, ethnographically-oriented research to better understand the socio-technical practices of deployment, development and implementation of FOSS systems in different ecosystem contexts — for example, FOSS systems for networked computer games versus scientific versus World-Wide Web applications. This is driven by the need to resolve the current paucity of detailed sociological research on this area of innovation, especially the absence of an account of mutual shaping between multiple constituencies (e.g., FOSS communities, corporations, governments) and emerging socio-technical dynamics in the collaborative development processes.

Evolution of Licensing Arrangements

Software property licenses are a relatively recent topic of research, and one that is becoming clearly relevant to studies of FOSS system development and evolution. Researchers have confirmed that the choice of a license correlates with some metrics of project success [Subramaniam, Sen, *et al.* 2009], although this research focuses on broad categories of licenses rather than specific licenses or specific license provisions. The task of determining which license applies to a specific built component or system has been addressed both by researchers [Gonzalez-Barahona, Robles, *et al*. 2009] and by industry projects [Gobeille 2008]. However they have noted that the binary files do not contain enough information to determine this completely.

License conflicts have been analyzed, with increasingly specific results, with no consideration of software context [Rosen 2004], minimal consideration of context

[German and Hassan 2009; Tuunanen, Koskinen, *et al.* 2009], and in terms of software architecture, build configuration, and version history [Alspaugh, Asuncion, *et al.* 2009a,b]. The analysis has proceeded from bottom up to identify or confirm the applicable license, starting with binary files [German and Hassan 2009; Gobeille 2008], and from the top down to guide design and assert a virtual license, starting with software architecture and other development artifacts [Alspaugh, Asuncion, *et al.* 2009a,b]. Such calculations are complicated by the fact that the license or license version for a specific component, or the interpretation of that license, can evolve over time. A recent study found that significant numbers of FOSS component licenses evolve across versions during the component's lifetime [Di Penta, German, *et al.* 2010]. The text and interpretations of licenses make it clear that license interactions depend at a minimum on how components are connected, and in some cases also on architecture, build, and executable run-time version data.

A palette of FOSS licenses exists, created and evolved to address goals, conflicts, and new technologies: wide use of academic software, an ever-increasing commons of FOSS, software patents and other IP, the rise of software services and embedded control software, and others. As new issues have arisen, new licenses (GPL, LGPL, MPL, EPL) and revisions of old licenses (GPLv3, Affero GPL) have appeared to address them. In the economy of license choices we see gradual shifts in the proportion of projects using various licenses, and occasionally projects changing from one version to another or one license to another.

How does the choice of license affect a FOSS project? There has been some work on how gross license characteristics correlate with certain project success measures [Stweart, Ammeter, et al., 2006; Subramaniam, Sen, *et al.* 2009]. However, there are other characteristics of interest such as flexibility in evolving the project's product and reliability, and the effect of licenses on these is not known. In addition, finer-grained license characteristics and individual license provisions are important as components of future revisions of existing licenses and of new licenses created to meet new goals. Subsequently, emerging research question include:

- How do software licenses shape how software is developed, evolved, used, and incorporated into larger systems?

- How does a license, and specific license provisions, affect significant FOSS product characteristics such as flexibility, reliability, interoperability, and degree of use?

- How does a license, and specific license provisions, affect significant FOSS project characteristics such as contributor population, development processes

and practices, coevolution with other projects, alignment with societal needs, and lifespan?

- How should license considerations be integrated into software development processes and tools?

Traditional FOSS projects are homogeneously licensed, with each project developed and released under a single license. An emerging context of FOSS development and use is for heterogeneously-licensed (HtL) systems, those composed of components developed by different projects and distributed under different FOSS and non-FOSS licenses. Here best-of-breed components are selected regardless of project or license, and integrated using shim code to quickly produce a complex system or systems with a very high degree of code reuse [Alspaugh, Asuncion, *et al.* 2009a]. The development context is now much wider, connecting a group of projects into a software ecosystem in which providers and consumers of software are connected by use and co-evolution relationships [Alspaugh, Asuncion, *et al.* 2009b]. Researchers also find that some traditional FOSS projects, when examined in detail, are found to be HtL due to apparent inadvertence [Gonzalez-Barahona, Robles, *et al* 2009]. The quality, prospects, and viability of an HtL system depend on the health of this ecosystem, and the increasing number of such systems leads to new goals for FOSS licenses. Other related research questions for this context include:

- What licenses and provisions support and encourage component reuse in HtL development, while still meeting existing FOSS goals?

- How can the rights and obligations within the virtual licenses of HtL systems be reliably managed by developers and satisfied by users?

- What effects do specific licenses and provisions have on the health of software ecosystems?

- In what ways should licenses evolve or be created to support software ecosystems?

- How do licenses interact with national and international law to have their proper effect?

Licenses address latent or expected conflicts between stakeholders. What is the role of license characteristics and provisions in mediating conflicts effectively, and guiding stakeholders in proactively forestalling potential conflicts?

Conclusions

Understanding the evolution of FOSS systems is a daunting yet fundamental problem for research. It is jointly a problem area for software engineering, FOSS development, human-centered computing, science studies, and the history of technology. The open, public availability of data about successful long-life FOSS systems represents a truly unique scientific opportunity to study the evolution of a complex systems technology. Such study is challenging in that the openness and availability of the information in no way trivializes the systematic and long-term effort necessary to fully comprehend how FOSS system evolution operates, and to what ends. Consequently, we need new ways to study FOSS system evolution so that we can acquire, articulate, and refine the scientific knowledge needed to explain, predict, and potentially control the evolution of FOSSD processes, collaborative work practices, project communities, project infrastructures and ecosystems. Future FOSS system research infrastructures will be critical to such studies of complex systems evolution.

We also need to expand our conceptual vocabulary beyond evolutionary metaphors drawn from classic Darwinian models of biological systems evolution. A richer set of constructs, metaphors, and relational models drawn from contemporary studies of biological, economic, cultural, and technological systems evolution need to be employed in explanations and of FOSS system evolution. The need is not simply to add more conceptual complexity to scientific discourse on FOSS but to recognize that the dominant framework for explaining, rationalizing, predicting, and controlling software systems evolution is still very limited after more than 30 years of study.

Finally, we serve the FOSS research community and the larger scientific research community well if we visualize the statics and dynamics of FOSS system evolution across releases, projects, and ecosystems as complex socio-technical systems. Evolution is not a condition of complex systems development; it is a web of processes that continually emerge and adapt across space, time, participants, geographical and cultural boundaries. Consequently, we should expect that our ability to explain, rationalize, or predict FOSS system evolution might more easily be done in textual, visual, and multi-media modalities using new tools and techniques created for such purposes [e.g., De Souza, Quirk, *et al*. 2007; Ogawa and Ma 2008; Ogawa, Ma, *et al.* 2008]. Such tools and techniques would likely be of significant scientific value to other research communities studying other kinds of natural or human-made complex systems.

Part III

FOSS Data, Analytics, and Research Infrastructure

A Research Infrastructure to Support New Science of Open Source Systems

Overview

Centralized, coordinated and collaborative research infrastructures are critical to allowing FOSS researchers to reach the high-impact future we envision([Gasser and Scacchi 2008]; [David and Spence 2003]; [Gasser, Ripoche, *et al.* 2004]; [Gasser and Scacchi 2003]).

The roadmap we are proposing for a new Science of Open Source Systems will necessarily include the need to continue developing a research infrastructure, which we describe in this chapter. As Gasser and Scacchi explain in [2008]:

> For F/OSS research, the objective [of research infrastructure] is to improve the collective productivity of software research by lowering the access cost and effort for data that will address the critical questions of software development research. ... In our view, the multi-discipline F/OSS research community seeks to establish a scholarly commons that provides for communicating, sharing, and building on the ideas, artifacts, tools, and facilities of community participants in an open, globally accessible, and public way.

Fortunately, there are a number of advantages FOSS researchers have in leveraging our own existing knowledge and efforts to build this research infrastructure commons.

First, evidence of the potential usefulness of a research infrastructure to FOSS research is already apparent and proven: several online research archives, tool repositories and community portals already exist to collect and promote use of these artifacts, e.g., FLOSSmole [Howison, Conklin, *et al.* 2006], SRDA [Van Antwerp and Madey 2008], FLOSSMetrics, FLOSSology, FLOSShub, etc. Numerous papers have been written using the shared data found in these basic community portals (examples: [Rossi, Russo, *et al.* 2010; Hofman and Reihle 2009; Wiggins, Howison *et al.* 2009; Squire and Duvall 2009; Crowston and Howison 2006; Ripoche and Sansonnet 2006]), and FOSS researchers have some record of sharing data and replicating each others' studies [English and Schweik 2007b; Wiggins and Crowston 2010]. These shared FOSS research repositories serve a critical role in sustaining the scientific basis for comparative FOSS system research studies. However, while these existing methods, tools, portals, repositories, and research archives each may have a small following,

none of them is really serving as an authoritative or centralized resource. The result is that researchers are often unaware of the multiple sources of online research data and tools; or, they may be dissuaded from using all those resources because of different interfaces and non-standard data formats; or data available on one resource may not be compatible with a tool on another resource site, etc. In addition, some of the data is replicated between data sources, and other potentially useful data is not yet collected and curated. This situation is not unique to research on FOSS; bioinformatics, environmental science, nanotechnology, scientists who study the effects of wind on structures, earthquake engineers, and many more have experienced these problems. Solutions to these problems in other domains often take the form of federated scientific portals (also called scientific gateways), enabled by emerging cyberinfrastructure technologies, designed and developed by a virtual organization of research and resource stakeholders. ([NSF 2007; Hey and Trefethen 2005; Freeman, Crawford, *et al.* 2005]) Such a standardized portal provides interoperability between users, data sources, and tools. A portal can also reduce the effort required to collect data and can result in better quality data with better annotations. [Parastatidis, Viegas, *et al.* 2009]

Second, we know that developing an infrastructure for research into the science of open source systems will be both accelerated and challenged by the very large amounts of data publicly available for study. This, as the saying goes, is both a blessing and a curse for FOSS research. These artifacts include software source code archives, archives of discussions between and among developers and users, statistics about projects (including usage, downloads, defects), social networking data (such as relationships between projects and developers, or between developers), characteristics of the code, descriptions of development processes, metrics describing success and failure of projects, models of various FOSS ecosystem components, documentation about projects or source code, software tools developed to research the FOSS phenomenon, and evidence of collaboration support (including written discussions and the like).

Third, the population of publicly accessible multi-project FOSS hosting repositories is growing in number, diversity, and projects hosted [COSSHF 2010]. SourceForge, Google Code, Codeplex, Freshmeat, Savannah, Apache, Tigris, and others host FOSS project content repositories, or links to separate project repositories. Some like Google Code, which may host upwards of 250K FOSS projects, and Microsoft's CodePlex are proprietary, and do not provide FOSS researchers with open access to their hosted projects databases nor allow offsite data collection programs to crawl and mine their contents, while other proprietary service vendors like SourceForge which also hosts about 250K FOSS projects do provide such access, and this access has enabled the creation of FOSS research repositories like SRDA and FLOSSmole. Others like Java.net and Tigris are specialized and only allow certain types of FOSS projects to be hosted. Still others like GitHub host nearly one million FOSS project directories, while having only 1/3 that number of users.

Third, other multi-project FOSS repositories are hosted by FOSS projects backed by nonprofit foundations like those for Savannah (Free Software Foundation) and Apache (Apache Foundation) but strive to ensure that hosted projects conform to certain guidelines, development practices, and software licenses. Development of FOSS research repositories that span and can grow to accommodate new multi-project FOSS project repositories is needed, as the kinds of data, meta-data, content links, and project content (online artifacts, source code bases, etc.) that these repositories host will continue to be both the source data and model for the development and use of physically decentralized, but logically centralized FOSS research infrastructures.

Finally, because we study the phenomenon of FOSS systems as collaboratively-developed, evolutionary software development ecosystems, we desire to transfer the benefits of such a system into our own research domain. The previous chapters on collaboration and evolution touched on some of the benefits from our research into systems developed in an “open” way. So far in our modest attempts to build a research infrastructure many researchers in our community have tried to “drink our own champagne” by applying principles of open, collaboratively-built systems to our own work. For example, the FLOSSmole and FLOSSMetrics projects collect and release data and scripts under an open source license; FLOSSmole, itself, uses a combination of features found within common software development forges.

It is our contention as researchers (and as users and leaders of these decentralized, nascent research portals) that a community-driven effort to coordinate our efforts and create a FOSS research infrastructure is critical to the success of our community's research agenda, as described earlier in this document. In this chapter, we describe what a FOSS research architecture would do for our research vision. We first present the possibilities of a FOSS research infrastructure. We then follow this with a description of each stage in our data collection and analysis process. In each stage of the process, we show the missing pieces in our current research infrastructure, and we outline how those might be addressed by a community-driven, federated FOSS research infrastructure.

Purpose of the New Infrastructure

A FOSS research infrastructure will provide a physically decentralized but logically centralized community for supporting research ideas with appropriate data and expertise. In addition, a common FOSS research infrastructure will make important contributions to non-FOSS research communities by providing standardized data sets to do replicable research (for example, by allowing empirical software engineering researchers access to very large data sets about FOSS development projects), or by

contributing to advance the state of the art in these other disciplines (for example, by allowing data mining researchers to apply their new algorithms to FOSS data sets, or by allowing database systems researchers to experiment with very large, heterogeneous data sets).

As discussed in more detail earlier in this report, research on the phenomenon of FOSS will have several high-impact benefits. It will 1) improve our understanding of how the FOSS ecosystem can be replicated in other domains, 2) provide insights on ways to improve innovation, education, and software engineering practices, and 3) contribute to the new science of open source systems. As with all research efforts, literature reviews are needed, studies must be designed and conducted, collaborations may be formed, data must be collected and analyzed, and results disseminated. The case for building shared resources to support these research activities is well established by organizations such as the NSF Office of Cyberinfrastructure and the many model research infrastructures already in place (some discussed below). Such an infrastructure will increase research productivity, reduce duplication of efforts, enable research not feasible today, and accelerate the dissemination of results. These benefits of the research infrastructure will be accrued for a variety of reasons. Having the data in a centralized, federated repository, researchers will be able to conduct unified searches across available data. Researchers can avoid duplication of data collection efforts, and will be less likely to abandon a research project because data is not easily available. Communications services (forums, email lists, wikis, informational web pages) and archival resources (document repositories, abstracts, bibliographies) will help researchers find collaborators, engage in Q & A with the research community, and discover related projects and results. Archived simulations, data mining tools, data manipulation and statistical scripts, stored queries, and other research support tools will help standardize and improve the productivity of FOSS researchers.

Examples of Research Infrastructures in Other Domains

Science-gateways (or portals) using sustainable cyberinfrastucture can promote distributed collaboration, increase research productivity, and enable research not feasible without such infrastructure. Examples from other research domains that serve as models for a FOSS research infrastructure include NEES, NEON, NCBI, and DataONE. NEES (Network for Earthquake Engineering Simulation) is a shared national network of experimental facilities, collaborative tools, communication forums, relevant news, and shared earthquake simulation software, a centralized data and document repository, and a Wiki. NEES enables communication and collaborative research and experiments by distributed researchers [NEES 2010]. NEON (National Ecological Observatory Network) collects data from multiple sites on the impacts of climate change, land use change and invasive species on natural resources and biodiversity

[NEON 2010]. NCBI (National Center for Biotechnology Information), run by the NIH, houses genomic and related data, an index of biomedical research articles, search engines, downloadable books, tutorials, and other related resources. VORTEX-Winds is a virtual organization of universities, institutes, firms, agencies and individuals that research methods to mitigate the effects of extreme winds on society; it pools geographically dispersed resources, centralizes collective knowledge, and serves as the virtual forum for the exchange of ideas to enhance the research and design capabilities of its members [Vortex-Winds 2010]. DataONE (Data Observation Network for Earth) is a gateway for accessing distributed environmental data available from atmospheric, ecological, hydrological, and oceanographic sources, serving scientists, land-managers, policy makers, students, educators, and the public through online access. [DataONE 2010] DataONE will ensure preservation and access to multi-scale, multi-discipline, and multi-national science data. These and many other gateways display what is needed in the FOSS research infrastructure: centralized access to distributed research resources. Such a science gateway could combine existing and new FOSS resources to provide similar features to those above for the FOSS research community.

Benefits of a FOSS Research Infrastructure

What data and tools will a FOSS research infrastructure include, and what will be the benefits of collecting and sharing these objects? As a start, Gasser and Scacchi [2008] outline five main *objects of study* in empirical FOSS research studies: software artifacts and source code, software processes, development projects, communities, and participants' knowledge. They have created a useful table — two columns of which we have reproduced here (see Table 4) — showing what each of these *objects of study* can tell us about "success" in software development. For the purposes of this document, we can consider each object shown in the table to be *type of data* to be collected, studied, analyzed, and discussed in our research infrastructure.

Objects	Success Measures
Source Code and Artifacts	Quality, downloads, reliability, usability, durability, fit, structure, growth, diversity, localization
Processes	Efficiency, ease of improvement, adaptability, effectiveness, complexity, manageability, predictability
Projects	Type, size, duration, number of participants, number of software versions released

Communities	Ease of creation, sustainability, trust, increased social capital, lower rate of participant turnover
Knowledge	Creation, codification, use, need, management

Table 4: Objects of study in a FOSS infrastructure, and what they tell us about success in software development [Gasser and Scacchi 2008].

A FOSS research infrastructure will enable more productive research into the new science of open source systems by encouraging researchers to collect, clean, analyze, and discuss these *objects of study*. In addition to the collection and curation of these objects of study, we also envision an infrastructure with tools and analyses, and a vibrant discussion portal. These are some of the benefits:

- *Eliminate redundancy*. Redundant efforts to collect and clean data will be reduced or eliminated. Data is easy to find. Data is trustworthy.

- *Exposed and explained*. Data, both basic and complex, is fully exposed for others to use. Conversations about the data are centralized and exposed, vibrant and timely. Site is easy-to-use.

- *Bootstrapping*. Analyses that are not possible today will become possible as research groups share their collective wisdom and enable each others' efforts.

- *Donations*. Data is donated and collected. Data sources are easy to annotate. Data sources and types grow and change over time.

- *Contributions to other domains*. The data, analysis, and discussions made possible by this research infrastructure will contribute to learning communities and open source development communities in other domains, e.g., open hardware design, open education, open science, open government, etc.

Building the Infrastructure

At the moment, the FOSS research infrastructure is very basic and underdeveloped. Researchers express interest in being able to store and share data of various levels of complexity, as shown in Table 1 above. These objects of study will consist of a variety of data types, from simple quantitative metadata about projects to complex workflows describing aggregation, cleaning, and analysis of data. The sections below follow a typical data analysis process, describing at each stage of that process what a FOSS

research infrastructure should include; what mechanisms are needed to facilitate that inclusion; what currently exists and whether it would meet the community's need if it were folded into a research infrastructure.

Data collection

After researchers identify the data they want to use in an analysis, the desired *data is collected* either manually or automatically. A research infrastructure should assist researchers with data collection in three ways. First, a research infrastructure should assist in the collection process through automatic data collection and centralized storage of commonly used data sets. Second, the research infrastructure should help the researcher find out what data has already been collected and understand the structure and provenance of the data that has been collected. Third, a research infrastructure should help researchers understand how to contribute their own data collections into the centralized storage. While the requirements for data collection may change over time, currently the most commonly requested data collection features for a centralized research infrastructure are:

- Storage of difficult-to-collect or very time-consuming-to-collect data types, including: data from source code forges, data from developer or user surveys, messages from project mailing lists, data from source code repositories (including version control information, dependency histories), data from issue trackers and bug reporting systems, developer and community discussions (emails, IRC, chat logs, blog posts and comments, forums, Twitter feeds), data about how users actually use the software, meeting or conference minutes or video streams.

- Privacy for community contributors, avoiding release of personal information (phone numbers, email addresses, employment information) or other sensitive data

- Regular and timely collection of data, performed automatically and exposed via easy-to-use methods.

- The ability to upload and share collected data, including data sets from published research papers or “Gold Standard” data sets.

Meeting these data collection needs will require a set of automated collectors for various data types, a centralized storage system (such as a database and file system),

a mechanism for creating downloadable data sets in many formats, a mechanism for uploading data sets, and a mechanism for discussing data sets between users.

Data curation and cleaning

Once the data has been collected, it needs to be described and stored in such a way that it can be shared and preserved over time. The features that FOSS researchers most frequently request for curated data are:

- A standard selection of just a few well-described, manageable formats for downloading data.
- A description of what has been “done” to the data, if anything, between when it was collected and when it was made available for use. Are there calculated fields? Are there interpolated values?
- A description of what data is missing in the collection.
- Grades or assessments of the quality of the collected data.
- The ability to upload and share data cleaning scripts.
- Assignment of other, well-described metadata to collected data.

Meeting these data curation and cleaning needs will require all the items from “data collection” above, plus a mechanism for describing the data (including tags), and a mechanism for downloading, uploading and sharing scripts that can manipulate the data.

Metadata

Metadata is data about the target data stored in the research infrastructure. For example, if a researcher is studying the concept of “a software project”, the data could include the source code about the project, whereas the project metadata could be high-level facts about that project, including its name, its creation date, or its URL. Or if a researcher is studying the behavior of a particular developer, the data could include the developer's messages to a project mailing list, whereas the metadata about that developer might be her name, her email address, her preferred spoken language, the list of projects to which she has contributed, etc. The metadata that will be stored in a

research infrastructure for FOSS will necessarily grow and change as the data stored in it grows and changes. At the moment, researchers commonly require the following:

- A standard language for describing key data elements, such as project, developer, user, action, fork, and release.

- The ability to tag data with metadata. Researchers typically want to be able to add contextual descriptors, such as intended user community, implementation language, and extent and nature of commercial support.

- Metadata that describes the provenance of the data. This may include validity assessments for the data, and these may be community-developed assessments.

- The ability to create and share connections between data sets. For example, information about how one project is similar to others.

- Other metadata that describes and fully documents the structure of the data.

Meeting these metadata needs will require all the items from the categories above, plus a mechanism for collaboratively writing and sharing a standard vocabulary, creating validity assessments, and drawing connections between the data sets.

Data analysis

A mature research infrastructure will provide more than just “data for download.” Ideally, a research infrastructure will also allow researchers to perform data analyses, and to contribute, describe, and comment on analyses performed externally. Some of the desired data analysis features include:

- The ability to automatically run various canned analyses on data collections already stored in the infrastructure.

- The ability to create and share constructed workflows describing analysis of data stored in the infrastructure (e.g. data about projects that has been collected).

- The ability to create and share aggregated or unified statistics from multiple data collections.

- The ability to create and share derived histories, for example constructing historical data from transactional databases, or re-constructing long-term histories of a project or of multiple projects.

- The ability to store interpretations or annotations of textual data, such as emails or IRC logs.

- The ability to store written case studies.

- The ability to store constructed taxonomies and ontologies.
- A set of tools that can be used online or downloaded that will perform common tasks. For example, source code control system analyzers, mailbox or mailing list analyzers.

Meeting these data analysis needs will require all the items from the categories above, plus a mechanism to select and run analysis tools, a mechanism to upload and share analysis artifacts (for example: workflows, case studies, ontologies), and a mechanism to upload and share user-developed software tools.

Using the data and talking about the data

Researchers have also requested a centralized, easy-to-use place to ask questions and discuss their data and findings with each other, including:

- What data exists? Where can I find it? Who contributed it? When? Is it accurate?

- How can I learn what other researchers have already done?

- What is the best application of my analysis to the Big Picture of FOSS research?

- How can I use FOSS research in my classroom? Can I contribute my own teaching materials to other researchers?

- I've recently completed this analysis. Is anyone in need of my expertise?

- Is there a common bibliography for FOSS work?

- Does anyone have archived copies of papers/articles/audio/video on some topic?

- I read this paper but I can't find the data/script/program. Does anyone know where I can get the data sets, programs or scripts that accompany papers?

- I want this new feature! (A structured feedback mechanism so researchers can help guide the actions of the infrastructure team.)

Meeting these needs will require all the items from the categories above, plus a mechanism to share bibliographical information, and a mechanism for tracking issues within the infrastructure itself.

Summary of Infrastructure Requirements

The following table summarized the infrastructure requirements described above. After the table, in section 3.7, each mechanism or requirement is further described, and we note any existing technology that can be leveraged for this infrastructure.

Required for Data Collection	Does it exist now?
1. Automated collectors for various data types	YES, but underdeveloped
2. Centralized storage system	YES, but underdeveloped
3. Mechanism for creating easily-downloadable data sets in multiple formats	YES, but underdeveloped
4. Mechanism for uploading data sets	YES, but not automated or described
5. Mechanism for discussing data sets between researchers	YES, but underdeveloped
Required for Data Curation	**Does it exist now?**
6. Mechanism for describing and tagging data	NO
Required for Creating Metadata	**Does it exist now?**
7. Standardized vocabulary for describing project metadata	YES, in theory, but not implemented, narrow focus
8. Mechanism for creating validity	NO

assessments	
9. Mechanism for drawing connections between data sets	YES, in theory, but not implemented, narrow focus
Required for Data Analysis	**Does it exist now?**
10. Mechanism to select and run analysis tools online	NO
11. Mechanism to upload and share analysis artifacts	NO
12. Mechanism to upload and share user-developed analysis tools	NO
Required for Using the Data	**Does it exist now?**
13. Mechanism to share bibliographical information	YES, but not collaborative, underdeveloped, underused
14. Mechanism for tracking issues in infrastructure	NO

Table 5: Infrastructure requirements to support the science of open source systems

Current Status of Infrastructure Requirements

Table 5 summarizes 14 requirements of a research infrastructure. These requirements are further explained in this section, along with the research tools that are being used currently, if any.

1. Automated collectors for various data types: a few repositories hold collections of research-oriented FOSS data. Most of the time, their data is first found at software forges (spidered or collected through web-based methods), cleaned, parsed, and stored in a central place for use by researchers. Some people have called these “repositories of repositories” (RoR). Examples are:

- *FLOSSmole* - many different forges. Metadata only.
- *FLOSSMetrics* - collects code and metadata about a hand-selected group of 1000 open source projects.
- *Notre Dame SourceForge Data (SRDA)* - Sourceforge data only.

- *Ohloh* - commercial implementation - non-research focus. Sourceforge data only. Basic metrics.

- *SPARS-J* - collects and searches through source code, not FOSS-specific. Basic metrics and code only, java, xml, and jsp.

- *Merobase* - collects and searches through source code and components, not FOSSspecific. Basic metrics. Java.

2. Centralized storage system: Several of the RoRs (namely FLOSSmole, FLOSSmetrics, SRDA) are specific to FOSS research and do contain a centralized, publicly-accessible database server.

3. Mechanism for creating easily-downloadable data sets in multiple formats. Most of the RoRs mentioned do provide mechanisms to select and download data sets in multiple formats. There is no general agreement about data set format, naming conventions, how files are described, their provenance, or whether they are verified.

4. Mechanism for uploading data sets. The sharing of data sets between researchers in a public way is currently a very rare occurrence in the FOSS community. When it does happen, it is a manual process involving project leaders.

5. Mechanism for discussing data sets between users. Currently, discussion of FOSS data sets takes place in three main venues: on the FLOSSmole mailing list, in personal communications between researchers, and at the annual OSS conference. A FOSS research infrastructure would provide a more robust means of fostering communication between researchers.

6. Mechanism for describing and tagging data. There are numerous data tagging systems in existence for other large, community-based data collection archives, but because of the decentralized nature of the FOSS research infrastructure, these have not been implemented. This would be a critical component of any FOSS research infrastructure.

7. Standardized vocabulary for describing project metadata. There are at least three standardized vocabularies which could be potentially useful in describing FOSS metadata, but none of these is complete. An effective strategy would likely embrace and extend one or more of these existing standards:

- *DOAP* - Description of a Project is an RDF description of the metadata describing any software project.

- *POM* - Project Object Model is designed for describing how to build a project within Apache Maven.

- *FAMIX* - is a metamodel for describing the static structure of any object-oriented software system.

8. Creating validity assessments for data sets. This does not currently exist conceptually, either in whole or in part.

9. Drawing connections between data sets. There are a few papers written about this possibility and some potential strategies for doing it, but nothing has been implemented.

10. Mechanism to select and run analysis tools online, live. Some tools exist for doing similar things. A sample of available tools includes:

- *CVSAnalY* is a simple tool used to extract data from CVS and Subversion version control systems. The tool reads the logs of a repository and recreates the data in a local database. It generates a few descriptive metrics, but is not generally extensible for other research. Researchers must pipe the output from CVSAnalY into some other tool to generate meaningful results. [Robles, Koch, *et al*. 2004]

- *FOSSology* is a data mining framework initially created as an internal project at Hewlett-Packard to identify software licenses present in source code [FOSSology 2010]. [Gobeille 2008]

- *Mylyn* is a plugin for Eclipse that allows researchers to use Eclipse to gather real-time data from developers. [Mylyn 2010]

- *Sourcerer* [Ossher, Bajracharya, *et al.* 2009; Bajracharya, Ossher, *et al.* 2009] is a set of possibly relevant open source tools. It includes a repository crawler for locating project metadata and download links, an automated dependency resolution tool for aiding compilation, a feature extractor for building relational models of (open) source code, and tools for building source code indices for searching.

- *Simal* is a framework for “the collection and management of essential details about open development projects” [Simal 2010].

- *Kepler* is a mature, open source workflow engine capable of “integrat[ing] disparate software components” [Kepler 2010]. In addition, Kepler workflows can be shared among collaborators. Kepler has many valuable features, including

broad support for distributed execution, graphical workflow creation, “a suite of data transformation actors”, and support for multiple programming languages.

- *Taverna* is an “application that eases the use and integration of the growing number of molecular biology tools and databases available on the web” [Hull, Wolstencroft, *et al.* 2006]. Although not created for FOSS research per se, it is a powerful workflow tool. Neither Taverna nor Kepler implements a storage medium in which to persist data.

- *Alitheia Core* is “an extensible software quality monitoring platform” [Gousios and Spinellis 2009] that implements an extensible pipeline for studying data from Subversion, MailDir, and Bugzilla. Its existence validates the need for a tool that manages both the logic and the data required for analysis. Much of the development overhead required to connect to repositories and to storage media, download and store the data, and store intermediate results is implemented by Alitheia Core. However, the framework does not provide extensibility for studying data from other types of repositories or sources (Git, Mercurial, Perforce, SourceSafe, GoogleCode, Trac, JIRA, etc.).

- *VisTrails* is “an open-source scientific workflow and provenance management system" developed at the University of Utah that provides support for data exploration and visualization” [VisTrails 2010].

11. Mechanism to upload and share analysis artifacts. Similar to #4 above.

12. Mechanism to upload and share user-developed analysis tools. Similar to #4 and #11 above.

13. Mechanism to upload and share bibliographical information: Currently there are a few project teams that are attempting to serve as *research hubs*. Research hubs are resources that attempt to collect research artifacts (papers, data sets) that have been produced by researchers about the open source software phenomenon. Examples of current FOSS research hubs are: FLOSShub.org. FLOSSmole has a rudimentary bibliography system to track its own data sets and to track papers that have been written using FLOSSmole data.

14. Mechanism for tracking issues within the infrastructure itself. There are numerous issue tracking systems in existence for other large, community-based software projects, but because of the decentralized nature of the FOSS research infrastructure, these have not been implemented. This would be a critical component of any FOSS research infrastructure.

Challenges for the FOSS research infrastructure

Challenges for the FOSS research infrastructure worth addressing at this juncture include:

- Quality. Assessing and monitoring the effects of this readily available data on the quality of the research that results. What is the best way to take donations and annotations while preserving quality and authenticity?

- Accessibility. Helping people find the data. Where do people look for data sets? Do they have to look very hard? What would it take to be a clearinghouse for data?

- Sharing. There are numerous challenges here:

- Privacy. Balancing researchers' needs for privacy and data protection against the advantages of data consolidation, collaboration, and open process in the research itself.

- Self-sufficiency. "Not invented here" syndrome. Students and faculty can be good at crunching out code and they like doing this. Encouraging researchers to trust and rely on data and code from other places can be a tough sell.

- Learning publicly. Difficulty in getting academics to learn in public. Acclimating researchers to the open, collaborative work process.

- Incentives. Do people assume that their formats and data won't be useful to others? Could donation requirements or incentives change this?

Conclusions

In this chapter we describe the gap between the research infrastructure that has been developed and what we still need in order to realize the high-impact future vision we articulate in previous chapters. FOSS systems have a transformative potential for many domains, and the vision we describe here of a centralized, coordinated, collaboratively-maintained research infrastructure will enable high-quality investigations of the five primary objects of FOSS study in the table above. The research infrastructure outlined here does not consist merely of storage of FOSS objects of study, but encompasses data collection, curation and cleansing, and analysis. With this infrastructure, we may

advance the science of FOSS in ways demonstrated by the research infrastructure of other domains. Finally, we believe the science enabled by this infrastructure will stimulate opportunities for growth in other science research programs and beyond.

Part IV

Broader Impacts of FOSS Research

Version of 29 November 2010

Broader Impacts Areas for Research in FOSS Systems

Overview

There are at least four major categories of broader impact arising from research in FOSS systems over the next 5-10 years. These are (a) software development, (b) education and learning, (c) innovation, and (d) science, industry, and government.

Software Development

The development of reliable large, very-large, or ultra-large scale software-intensive systems requires more than robust, formalized, and mathematically grounded approaches to software engineering. They also require the engagement of decentralized communities of practitioners who can participate in and contribute to the ongoing development, use, and evolution of software system tools, online artifacts, and other information infrastructure resources, either on a local or global basis. The development of software-intensive systems at large-scale and beyond needs to be recognized as something now essential to the advancement of science, technology, industry, government, and society across geographic borders and cultural boundaries.

FOSS systems research is likely to change how software engineering research and practice are now accomplished. The openness of FOSS system development means that new participants are coming into the world of software systems to contribute to their development and evolution. The engagement and contribution of participants who are not necessarily skilled in the traditional principles and practices of SE means there will be a long-term need to adapt SE concepts, techniques and tools to people lacking skills in SE, while also seeking ways for motivating these new participants to engage in learning and practicing emerging SE processes, practices, and principles. In addition, the public availability of FOSSD artifacts will likely become a primarily source of data for empirical SE research, as such data will often be far less encumbered by the corporate non-disclosure agreements that have historically limited what software development data can be made available for scientific research purposes.

FOSS systems research will continue to be a rich source of observation and experimentation for collaborative software development processes, practices, and project forms. As many successful, ongoing, and large-scale FOSS systems and project

communities are typically physically decentralized but logically centralized, sustained software development must rely on collaboration tools, techniques, and patterns of use whose fundamental principles we do yet fully understand. Yet, FOSS system development is a clear, recurring demonstration that the development of complex systems can be performed, governed, and sustained in a decentralized manner, with little or none of the corporate oversight or enterprise governance that have long been a hallmark for the development and maintenance large complex systems. Collaborative FOSS system development processes, practices, project forms, project infrastructures, and surrounding ecosystem represent new ways and means for developing complex systems that meet societal needs.

FOSS systems depend on and co-evolve with their surrounding ecosystem. They are both social and technological endeavors, in which socio-technical interactions are more critical to system development, use, and evolution than a formal mathematical basis for specifying the system's analytical intent. The study of FOSS system ecosystems is at a very early stage. But human-made complex systems are increasingly recognized as being products of their own complex ecosystems, and of the networks of producer, integrators, and consumers who create, assemble, and use such systems. Thus, research into complex system ecosystems like those that situate and embed FOSS systems are within the grasp of scientific study, comprehension, and explanation. These eventual accomplishments will provide the basis for rationalizing, predicting, controlling, and transferring such knowledge to other complex ecosystems, especially those that are mediated by information infrastructures or cyberinfrastructure. Thus research into FOSS ecosystems is critical to advancing scientific knowledge and technology development in many areas beyond software systems.

FOSS systems are complex software systems with an open evolutionary history and future. Such openness is in many ways historically unprecedented for complex technical systems. So we should not miss a rare opportunity to study FOSS system and ecosystem evolution, as a software system, as a decentralized social system for peer production, and as a complex socio-technical system.

Education and Learning

We need to educate a new generation of students and other publics to understand how best to create, access, study, modify, and share complex systems that are open and liberating. This requires widespread information resources, development processes, work practices, and online content/assets that are free and open, rather than costly, opaque, and restricted to those who can afford to access them. These will provide the new baseline for transforming education and learning in the sciences, industry, and democratic government.

Software engineering education is a prime target for adoption of FOSS system development processes, practices, and project forms. Other CISE disciplines may also benefit from introduction and integration of openness in team-oriented project work courses. Open source projects are a viable conduit for encouraging collaborative study, practice, experimentation, and sharing. FOSSD projects are excellent vehicles for creating, extending, and sharing knowledge about large-scale, complex software systems in ways that can create new workforces where they don't exist, or where they are underdeveloped. To the extent that the socio-economic growth of scientific fields of study, national industries and public institutions are increasingly dependent on larger and complex software systems and infrastructures, then fostering and stimulating FOSS system research, practice, and education will be a critical national investment.

However, FOSS system development is not a complete remedy for education and learning in CISE coursework. FOSSD projects are often skewed culturally and by gender (far too few women are visible contributors to FOSSD projects), and why this is so is not well understood. FOSSD projects are often globally dispersed, but such dispersion is not uniformly distributed across all countries, ethnicities, or cultures. There is still much to learn about how to stimulate the benefits of FOSSD projects and other open practices in ways that are inclusive and that mitigate the barriers to participation and contribution that seem to exist, in spite of the openness at hand.

In sum, FOSS system development will facilitate new ways and means for education and learning in CISE coursework and real-world practice. But it is not a panacea that will remedy or overcome the barriers to participation that at present remain poorly understood.

Innovation

Engines of innovation for advancing science, technology, and engineering in industry, government, and society at large are few and far between. FOSS systems development is emerging as one such engine whose openness encourages invention and reinvention, knowledge sharing and crowd-sourcing, and lower cost access to higher capability information technologies. Further, these technologies are transparent and open for widespread public access, study, modification, experimentation, ad hoc or systematic integration, repackaging, and redistribution. FOSS systems can stimulate societal advances, innovations, and progressive transformations when their openness is assured and protected.

Many FOSS systems and their development projects are berated as efforts to merely copy existing commercial software system products. But this perspective undervalues

the effort of reinvention as a critical innovation practice. Reinvention is a fundamental strategy for learning how successful products are made. Efforts to discover what is going on inside of “black boxes” is a widespread practice of curiosity and inquiry. This kind of endeavor can serve as the basis for improving or making entirely new products. Reinvention and rediscovery is also one of the most common approaches to education in science, engineering, technological, and mathematical coursework. So FOSS systems and projects that recreate successful complex systems are a vital component of workforce and socio-economic development. Such practice should be encouraged and celebrated as a national strategy that fosters technological progress and societal advancement.

FOSSD projects generally seem to practice open innovation that is user led. Participatory or democratic innovation [von Hippel 2001, 2006] is a new mode of technological innovation. It is also a hallmark of an advanced society where progress is assured through democratic participation in complex systems development. Participants can get involved in innovation practices as end-users who identify potential FOSS system weaknesses. These simple contributions can lead to open pathways to more opportunity to affect or control how a FOSS system continues to develop and evolve on its way to becoming evermore useful. But progressing along these pathways will not be for the faint of heart, as the challenges to be met will often co-mingle social and technical relationships. This is where the secret sauce of innovation may be found—in learning how to navigate such relationships to achieve a vision with the least amount of time and effort. The openness of FOSSD projects can become an engine of innovation for new participants who find their way through their web of socio-technical challenges.

Science, Industry, and Government

Many grand challenges for science and engineering depend on the research and development of a new generation of complex, software-intensive systems. Advanced healthcare informatics, advanced personalized learning systems, secure cyberspace, engineering automated tools for scientific discovery, and enhanced virtual reality are all readily recognized as problem domains that depend on future software systems. Making solar energy economical, managing the nitrogen cycle, preventing nuclear terror, providing energy from fusion and access to clean water, engineering better medicines, developing carbon sequestration methods, improving urban infrastructure, and reverse engineering the human brain are also areas where new generations of software systems are needed to enable and deploy the sought after scientific advances. But meeting these grand challenges depends on more than robust or well-engineered software systems. They will also depend in part or full on FOSS systems and ecosystems, as well as FOSSD processes, practices, and project forms.

Social choices and economic constraints may make proprietary or closed source system solutions less practical and less desirable. For example, scientific research into fusion energy centers around the International Thermonuclear Energy Research (ITER) project, still in the early stages of development, has a budget forecast at more than $20B. How much of that budget will be allocated to development of ITER control system software, and who will be called upon to develop or engineer the requisite software? ITER is a multi-national effort, and there is likely to be a common call for openness in its software development projects, as well as openness in science practices, rather than an expectation that some company or contractor will develop a proprietary, closed source software system. As such, it may be the case that grand challenge problems are more likely to embrace or demand openness in their system development efforts, at least prior to any commercialization of supporting software systems.

FOSS system development has already begun to transform the global software industry and all major software and Information Technology (IT) firms. Proprietary, closed source systems are not likely to disappear, but there will be growing pressure on proprietary systems to offer innovative features or functions that are not yet available as FOSS systems. FOSS systems may therefore motivate proprietary system developers to advance technologically out of self interest and preservation of market position. Once again, FOSS systems are creative drivers that stimulate advances to the broader economy and IT marketplace.

Companies that actively resist the progressive transition to FOSS systems will be increasingly marginalized. FOSS systems will take over mundane, infrastructural, and non-competitive IT domains, and this will help to clarify where IT or software system value truly is to be found. Stimulating research and development into FOSS systems and FOSSD projects are a strategic national investment, if the goal is to improve national and industrial IT system capabilities and related industries.

Advances in enterprise information systems that streamline operations, create new products or services, more stimulating jobs, and workforce development opportunities, depend on faster, better, and cheaper software systems. Helping to make regional and national governments more transparent, open, and trustworthy requires public access to information systems that are easy to access, open for study and open to citizen participation. FOSS systems are the most likely technology to meet these societal needs.

Recommendations for Action

FOSS system development (FOSSD) is emerging as an alternative approach to developing large software systems. FOSSD employs socio-technical development processes, work practices, and networked community project forms. These processes, practices, and project forms often differ from those found in industrial software projects, and those portrayed in software engineering (SE) textbooks. As a result, FOSSD offers new kinds of processes, practices and project organization. Understanding and explaining how FOSSD processes, practices and projects are similar to or different from their traditional SE counterparts is an area ripe for further research and comparative study. This includes understanding how and why collaboration works within FOSSD projects, how FOSS systems are situated within software ecosystems, and how FOSS system evolve. The studies reviewed and research questions identified throughout this report lead us to several recommendations for facilitating a new science of openness.

The recommendations we put forward seek to realize a discontinuous leap in the current state of scientific knowledge in Computer and Information Science and Engineering (CISE) research efforts. Their collective goal is to identify how the development of FOSS systems can enable significant constructive transformations, as well as broader societal benefits and impacts. These recommendations follow and complete this report.

Recommendation 1: Stimulate investment in projects for scientific research and technology development that build FOSS systems as a way to stimulate workforce development.

The creation of a new, skilled and motivated workforce is not inevitable from any research program, unless it is designed toward such an outcome. FOSSD poses this opportunity, as witnessed by large-scale demonstrations orchestrated by companies like Google and its Summer of Code projects that engage thousands of students world-wide each year, with comparatively high levels of successful and deployed student projects. More broadly, hundreds of thousands of self-initiated FOSS projects seek to create and deploy FOSS systems. The vast majority of these projects fail. Yet they embody intrinsic motivations on the part of their participants to learn how to build and use such systems, whether for personal, academic, industrial, or governmental application.

FOSSD is already beginning to transform software and IT industries, as well as scientific research projects that need software-intensive systems for data collection, analysis,

networking, visualization, and dissemination. Similarly, the adoption of FOSS systems within academia, for-profit and non-profit enterprises, and government agencies at the local, regional, national, and international levels will create a workforce that is motivated and skilled in new FOSS-based concepts, techniques, and tools grounded in CISE disciplines.
The integration of FOSS systems into CISE education will bring new participants into the broader field of study and practice, as well as heighten awareness of the value of being a contributor to FOSS system development projects. .

Recommendation 2: Create a new cross-cutting research program within the CISE Directorate that supports all aspects of FOSS systems research—FOSS development processes, work practices, and alternative project forms; collaboration in development and use of FOSS systems; FOSS ecosystems; and FOSS system evolution.

The bulk of this report identifies FOSS studies conducted within the last ten years, along with emerging research problems or questions. However, the creation of new scientific knowledge in FOSS systems and development is spread among diverse CISE researchers and research programs that don't seem to be well connected. Perhaps this is inevitable, desirable or both. But the recommendation here is strategic--it seeks to improve the effectiveness of investments in FOSS systems research by bringing isolated studies together into a coherent community.

FOSS system research spans many diverse disciplines, both within and beyond CISE disciplines. Few advances in CISE fields have garnered such widespread interest. When they do attract attention, they become transformative and enable broad impacts. Once again, the Internet and World Wide Web are prime examples of complex systems whose core software relies on FOSS systems, as well as the socio-technical projects and ecosystems that evolve them. Should they be left alone to fend for themselves? Why not consider how a coherent programmatic organization could focus scarce attention and limited resources to FOSS system research and enable many desirable transformations and broader impacts that have been identified through this report? Not merely a recommendation to reorganize the management of FOSS systems research, this is a call to arms in the national interest, as well as in the interest of researchers and practitioners across the CISE disciplines.

However, if available resources for new research programs is limited or creation of a new research program unrealistic, then at minimum, it is appropriate to invest in a national center for research in the science and technology of FOSS systems.

Recommendation 3: Stimulate research in development and use of FOSS systems in other science research programs, health, energy, climate, defense, and National Engineering Challenge domains.

In concert with the two preceding recommendations, there is a strategic opportunity to stimulate and invest in FOSS systems research to create new scientific knowledge in other non-CISE disciplines. The majority of National Engineering Challenges for the 21st Century found in the National Academies report on the subject depend on software systems at their core. These systems will not be developed by computer scientists or software engineers, but by scientists and engineers working within their specialities in health, energy, climate and other domains.

An investment to stimulate the research and development of FOSS systems can simultaneously be an investment to advance R&D in these National Challenge domains. The development processes, work practices, and alternative project forms associated with successful, self-organized, and self-governed FOSSD projects are likely those needed by R&D projects in the Challenge domains. Similarly, the collaboration patterns, software ecosystems, and evolution processes found in self-sustaining FOSS system development projects are also those needed in the Challenge domains. Effort in this area may or may not be redundant with the previous recommendation, depending on how the new research program or office is structured and administratively located. This recommended action is directed to support the scientific research community outside of the CISE disciplines, much like the Office of Cyberinfrastructure or other NSF-wide Cross-Cutting Program do. Why? Part of the answer stems from growing recognition that science researchers outside of the CISE directorate are already investing in the development and use of software-intensive systems that rely on FOSS systems at their core. FOSS systems are becoming an ever more central element of the infrastructures of science, and thus participate in, mediate, or enable scientific discoveries and advances in many disciplines.

FOSS systems and their development are becoming a strategic capability for advancing all of science, not just software engineering or human-centered computing. The software systems that make up the Internet and World-Wide Web are based on FOSS systems and FOSSD projects, as are much of global e-Commerce infrastructure, email, Web search, and other Web generation enterprises and R&D projects. Yet, much of the science research community has not followed FOSS community practices like the establishment of publicly accessible, shared repositories of software, online artifacts, data, and publications that enable curious people to browse, use, or contribute to such scientific research or technology development projects.

An investment in research on open source science and FOSS systems is a strategic one that is best viewed as Pareto optimal, rather than as zero sum, across science and engineering disciplines referenced by the National Engineering Challenges.

Recommendation 4: Stimulate research in Gender and FOSS, and Collaboration and Diversity in FOSSD.

Creating FOSS systems and FOSS development projects is technically challenging work that can directly benefit from results and findings in software engineering and human-centered computing research studies. As already noted, the vast majority of FOSSD projects fail to develop and deploy usable systems. This is fine as long as modest resources go into starting new FOSSD projects. But it is a risky venture for those not already skilled and knowledgeable about FOSSD processes, practices, and project forms to invest scarce resources to build systems that may ultimately lack the critical socio-technical mass to succeed, grow, and evolve.

There are some poignant gaps in how FOSSD processes, practices, and projects work, whether local or global in scale and participation. For example, why are there still comparatively few women involved in FOSSD projects? Further, it may also be the case that proportionally, there are fewer women involved in FOSSD than compared to academic CISE disciplines or the broader software/IT industries. Why is this so? Is openness somehow an issue of gender, one that is tilted towards males? Is FOSSD participatory, contributory, and successful only when a largely homogeneous community of like-minded developer-users are involved? Does FOSSD implicitly encourage some form of meritocratic exclusion across (a) all groups, (b) people with certain characteristics, or (c) gender? Do meritocratic forms of FOSSD governance enable certain subtle forms of discrimination?

There is much to be inspired and motivated by FOSSD projects and outcomes, but is limited diversity and gender imbalance inevitable, probable, temporary or accidental?

Similar questions can be raised about collaboration across socially and culturally diverse communities. Though FOSSD is often cited as a model of collaborative practice on a global basis, FOSSD projects tend not to be fully or uniformly global, engaging people from all countries interested in participating. What is needed for FOSS systems or projects to become universal?

As these questions have not been well addressed, it seems reasonable to recommend that whatever form or initiative is taken to invest in building up a new cross-cutting

program or office for research supporting openness and FOSS science, it is strongly advised that these current blind spots in the research need to be recognized and addressed programmatically. If increased gender balance and diversity would thrive in a world of openness or open source systems, what can be done to create such a world in a timely manner?

Recommendation 5: Invest in and encourage cross-cultural studies of FOSS, especially in non-English cultures.

Once again, as related and informed by the preceding recommendation, much of what we know about FOSS systems and FOSSD projects comes from studies of those of English-based communities. However we have very little scientific knowledge about whether or how work, community, or collaborative practices in non-English based FOSSD projects operate. SourceForge Japan, for example, hosts many thousands of FOSSD projects that are openly accessible to those fluent in Japanese, but not in English. What is going on in these projects? Do they replicate and naturally reproduce the same kinds of development processes, work practices, alternative project forms, collaboration patterns, and system evolution processes that we see in U.S., European, or other Western-based FOSSD projects conducted in English? If so, this would suggest cultural differences matter less than technical skill and commitment to learn through participation. If not, perhaps different cultures cherish and enact different forms of collaboration or meritocracy, or may find constructive social value in other choices for how best to develop, deploy and sustain complex software systems.

In non-English or non-Western based societies, under what conditions will software developers embrace openness in complex systems development? Is openness predominantly a Western and English-based cultural value? Is “freedom enabling” software development seen as a subversive or hegemonic ethical value, or is it a global social movement for cultural transformation?

Recommendation 6: Stimulate the research and development of FOSS systems for humanitarian aid and relief, especially those that provide opportunities for graduate, undergraduate, and secondary students to contribute.

One promising area of effort to develop and deploy FOSS systems is that focusing on humanitarian aid and relief across geographic borders. As natural disasters and human-led disasters continue to arise in seemingly more complex forms, it appears that FOSS-based systems and ecosystems are emerging to create or address new ways to reduce

the associated human suffering and loss. FOSS systems for humanitarian aid and relief have been deployed in places like Haiti, following the recent earthquake disaster, to help locate displaced family members, as well as to keep track of what relief resources have been deployed and where they have been distributed. S.

Once again, an important characteristic of many successful FOSSD projects is their ability to encourage volunteers to contribute to ongoing system improvement and evolution. Such contributions do not necessarily require high levels of core software development effort and skill, though they do represent more than casual experience as an end-user of such systems. Even modest participation can open windows of awareness that cross cultural divides and enable new modes of societal engagement, at a time when other modes of assistance seem distant, institutionally diffuse or opaque. Furthermore, such systems are situated in FOSSD ecosystems that enable both CISE and non-CISE students to participate and get involved in ways that traditional coursework may not have allowed.

Recommendation 7: Stimulate existing research programs in Software Engineering, Human-Centered Computing, and Networking Technology and Systems to investigate and develop new approaches to the challenges of engineering FOSS systems and real-world systems that rely of FOSS.

As indicated in the preceding recommendations, research and development of FOSS systems are likely to yield broader impacts and socio-economic benefits. However, existing research programs in Software Engineering (Computer and Communications Foundations Division), Human-Centered Computing (Information & Intelligent Systems Division), and Networking Technology and Systems (Computer and Network Systems) are still needed, as the existence and pervasive distribution of FOSS systems exacerbates some of the research problems that are emerging in studies in these respective programs. For example, academic research in software engineering historically has been limited to study of software systems that could be prototyped in laboratory settings, or to conduct of empirical studies of software development efforts *in situ* that may be subject to access and disclosure limits. The widespread growth, diffusion, and availability of FOSS systems changes this, by providing access to open and operational (or in development) systems that may have independent developer-user project communities, where the FOSS systems, development artifacts, and project social networks are also open for access and study.

Research addressing the composition of large or very large-scale software systems that benefit from new specification/design languages, notational techniques or quality testing tools, can now often be built and demonstrated at scale using FOSS systems as their

components, or objects of study. Building software systems including more than 1 million lines of source code within an academic setting was almost unimaginable a generation ago, yet it is increasingly common among researchers who work with FOSS systems It also means that problems of engineering software systems of this scale and complexity can now be studied, both as experiments in new ways and means to build such systems, and as socio-technical objects of empirical study within their embedding *in vivo* ecosystem or isolated *in vitro* laboratory setting. However, what happens when large or very large-scale FOSS systems are built from diffuse aggregate collections of software components developed by new developers with little or no expertise in software engineering principles? The precarious creation and deployment of such systems poses new challenges for how to sustain and systematically evolve them for software engineers and human-centered computing scientists.

As new software engineering, human-centered computing, and networking systems are built using FOSS systems as elements, both outstanding and new research problems are manifested. FOSS systems are helping to advance the creation and refinement of new scientific knowledge in these areas. As noted in the broader impacts section in this report, FOSS systems are an engine of innovation that can drive scientific and socio-technical advances in software engineering, education, science, industry and government. So it is important to recognize that an investment in FOSS systems and openness should not be considered as a reallocation of resources away from current CISE research programs, but one that strategically advances research and scientific knowledge in these areas. A strategic investment in open source science and FOSS systems research and development is one that is most effective if complementary rather than if zero-sum.

Recommendation 8: Establish and support shared research repositories for FOSS data as part of the new research infrastructure.

FOSSD project source code, artifacts, and online repositories offer new publicly available data sources of a size, diversity, and complexity not previously available for SE research, on a global basis. The current FOSS research infrastructure is modest and relies on donations of FOSS data from projects. This offers an unnecessarily limited perspective of a select set of projects. Expansion of research infrastructure will support additional kinds of data from a nonrestrictive project sets as a vital basis for further research and development of FOSS science. Similarly, investment in “repositories of repositories” will enable scientific research and knowledge creation in FOSS systems to scale from small to very large-scale studies.

Recommendation 9: Pursue development of advanced data analysis tools for examining FOSS data as part of the new FOSS systems research infrastructure.

Many new research opportunities exist in the empirical examination, modeling, and simulation of FOSSD activities and communities. We cannot predict when, where, why, how, or with whom FOSSD projects will work effectively or efficiently. Similarly, we lack the scientific knowledge needed to explain how FOSS systems evolve over time, or within or across different software ecosystems. The popularity and unbridled enthusiasm of the thousands of young software developers who avidly participate in and contribute to FOSSD projects indicates that FOSSD processes, practices, and projects are being diffused, adopted, adapted, and transferred in interesting ways. We lack the scientific knowledge to explain why this is happening, and with what consequences. FOSS research, at present, suffers from a tremendous lack of data and a dearth of tools to analyze and understand the FOSS phenomenon. Those tools that are available are still rudimentary and are mostly limited to supporting only quantitative data analyses. Tools and techniques that discover, extract, analyze, model, or visualize qualitative data or processes are nascent but needed to produce new scientific knowledge about FOSS systems and socio-technical resource arrangements.

Overall, these nine recommendations represent our collective position on how to achieve the most significant scientific results, to create necessary, new scientific knowledge, to transform academic, industrial and government R&D efforts, and to unleash the engine of innovation that FOSS systems can enable and fuel.

Contributors

This report is the result of a workshop held from 10-12 February 2010 in Irvine/Newport Beach, California and a follow-up meeting from 22-24 June 2010 in Evanston, Illinois regarding the Future of Free and Open Source Software. The workshop was organized by Walt Scacchi, Institute for Software Research, University of California, Irvine; Kevin Crowston, Syracuse University School of Information Studies; Greg Madey, University of Notre Dame; and Megan Squire, Elon University. The report reflects the input from forty-five participants, listed below, from the research community, including academia, industry, and members of the open source community:

Mark Ackerman, University of Michigan, Ann Arbor
Thomas Alspaugh, Institute for Software Research, University of California, Irvine
Paula Bach, Microsoft Corporation
Sushil Bajracharya, University of California, Irvine
Shobha Chengalur-Smith, University at Albany, SUNY
Premkumar Devanbu, University of California, Davis
Hamid Ekbia, Indiana University
Justin Erenkrantz, The Apache Software Foundation
Roy T. Fielding, Day Software
Les Gasser, Graduate School of Library and Information Science, University of Illinois, Urbana-Champaign
Daniel German, University of Victoria
Robert Gobeille, Hewlett Packard
Martina Greiner, University of Nebraska at Omaha
Imed Hammouda, Tampere University of Technology
Scott Hissam, Carnegie Mellon Software Engineering Institute
James Howison, Carnegie Mellon University
Chris Jensen, Institute for Software Research, University of California, Irvine
Yuzo Kanomata, Institute for Software Research, University of California, Irvine
Christopher Kelty, Department of Information Studies, University of California, Los Angeles
John Leslie King, School of Information, University of Michigan, Ann Arbor
Charles Knutson, Brigham Young University
Cristina Lopes, Institute for Software Research, University of California, Irvine
Kalle Lyytinen, Case Western Reserve University
Alex MacLean, Brigham Young University
Audris Mockus, Avaya Labs Research
Ralph Morelli, Trinity College
Luis Felipe Rosado Murillo, Department of Information Studies, University of California,

Los Angeles
John Noll, Lero, The Irish Software Engineering Research Centre
Joel Ossher, University of California, Irvine
Stormy Peters, GNOME Foundation
Jack Repenning, CollabNet, Inc.
John Riedl, University of Minnesota
Dirk Riehle, Friedrich-Alexander-University of Erlangen-Nürnberg
Jason Robbins, Google Inc.
Charlie Schweik, Dept of Natural Resources Conservation and Center for Public Policy and Administration, University of Massachusetts, Amherst
Susan Sim, Institute for Software Research, University of California, Irvine
Michael Twidale, Graduate School of Library and Information Science, University of Illinois, Urbana-Champaign
Patrick Wagstrom, IBM Research
Tony Wasserman, Carnegie Mellon Silicon Valley
Kangning Wei, Syracuse University
Joel West, San Jose State University
Hyrum Wright, Subversion Corporation

Acknowledgements

Administrative support for the workshop came from Debra Brodbeck and Kiana Fallah. Technical and Web services support came from Yuzo Kanomata and Kari Nies. All are from the Institute for Software Research at the University of California, Irvine.

Last, the *2010 Workshop on the Future of Research in Free/Open Source Software*, 10-12 February 2010, was supported by a grant from the Computing Community Consortium to the Institute for Software Research http://www.isr.uci.edu/ at the University of California, Irvine. Support for Scacchi, Jensen and others at the UCI Institute for Software Research is also provided in part from the National Science Foundation grant #0808783. No review, approval or endorsement is implied. Further information about the Workshop can be found at: http://foss2010.isr.uci.edu/.

References

[2020FLOSS 2010] *2020 FLOSS Roadmap*. http://www.2020flossroadmap.org/download/ and http://www.2020flossroadmap.org/wp-content/uploads/2010/09/2020-FLOSS-Roadmap-2010-Version-1.2.pdf, last accessed 30 September 2010.

[Agerfalk and Fitzgerald 2008] Agerfalk, P. and Fitzgerald, B. (2008). Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy. *MIS Quarterly*, 32(2): 385-410.

[Aksulu and Wade 2010] Askulu, A. and Wade, M. (2010). A Comprehensive Review and Synthesis of Open Source Research, *J. Association for Information Systems*, 11(11), 576-656. November.

[Ajila and Wu 2007] Ajila, S. A. and Wu, D. (2007). Empirical study of the effects of open source adoption on software development economics. *The Journal of Systems and Software*, 80(9): 1517-1529.

[Alberts and Hayes 2003] Alberts, D. S. and Hayes, R. E. (2003). *Power to the Edge: Command and Control in the Information Age.* CCRP Publications, Washington, D.C., DoD Command and Control Research Program. (available at http://www.dodccrp.org/files/Alberts_Power.pdf)

[Aldea, Jones, *et al*. 2003] Aldea, F., Jones, M., and Schabe, H. (2003). Checking Whether SCOS is up to SPEC. *ESA Bulletin 115*: 58-60.

[Allison, Currall, *et al*. 2005] Allison, A., Currall, J., Moss, M., and Stuart, S. (2005). Digital identity matters. *Journal of the American Society for Information Science and Technology*, 56(4): 364–372.

[Alspaugh, Asuncion, *et al.* 2009a] Alspaugh, T. A., Asuncion, H. U., and Scacchi, W. (2009). Intellectual Property Rights Requirements for Heterogeneously-Licensed Systems. *International Requirements Engineering Conference (RE'09),* Atlanta, GA: 24-33.

[Alspaugh, Asuncion, *et al.* 2009b] Alspaugh, T. A., Asuncion, H. U., and Scacchi, W. (2009). The Role of Software Licenses in Open Architecture Ecosystems. *International Workshop on Software Ecosystems (IWSECO 2009)*, Falls Church, VA: 4-18.

[Alter 1999] Alter, S. (1999). *Information Systems, A Management Perspective*. Third Edition. Addison-Wesley, Reading, MA.

[Andrews 1991] Andrews, G. R. (1991) Paradigms for process interaction in distributed programs. *ACM Computing Surveys*, ACM Press, 1991, 23(1), 49-90

[Atkinson, Weeks, *et al.* 2004] Atkinson, D. C., Weeks, D. C., and Noll J. (2004). The design of evolutionary process modeling languages. *11th Asia-Pacific Software Engineering Conference*, Busan, Korea: 587–592.

[Augustin, Bressler, *et al*. 2002] Augustin, L., Bressler, D., and Smith, G. (2002). Accelerating Software Development through Collaboration. *24th International Conference Software Engineering (ICSE 2002)*, Orlando, FL: 559-563.

[Augustin 2010] Augstin, L. (2010). Commercial Open Source for the Enterprise: Plenary Address. *6th IFIP International Conference on Open Source Systems*, Notre Dame University, South Bend, IN.

[Bajracharya, Ossher, *et al.* 2009] Bajracharya, S., Ossher, J., and Lopes, C. (2009) Sourcerer: An Internet-Scale Software Repository. *SUITE 2009: First International Workshop on Search-driven Development – Users, Infrastructure, Tools and Evaluation* (co-located with ICSE 2009), Vancouver, B.C., Canada: 1-4.

[Baldwin and Clark 2003] Baldwin, C. Y. and Clark, K. B. (2003) The Architecture of Cooperation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model? *Harvard Business School Working Paper*, No. 03-209, 03(209): 1-63.

[Baldwin and von Hippel 2009] Baldwin, C. Y. and von Hippel, E. A. (2009). Modeling a Paradigm Shift: From Producer Innovation to User and Open Collaborative Innovation. *Harvard Business School Working Paper*, No. 10-038, 10(038): 1-36.

[Balka, Raasch, *et al*. 2009] Balka, K., Raasch, C., and Herstatt, C. (2009) Open source enters the world of atoms: A statistical analysis of open design. *First Monday,* 14(11).

[Battilana 1995] Battilana, M. C. (1995). The GIF Controversy: A Software Developer's Perspective. Last revision June 20, 2004. Original text published January 27, 1995. Last accessed 30 June 2010, from http://www.cloanto.com/users/mcb/19950127giflzw.html

[Becerra-Fernandez, Elam, *et al.* 2010] Becerra-Fernandez, I., Elam, J., and Clemmons, S. (2010). Reversing the landslide in computer-related degree programs. *Communications of the ACM*, vol 53 (2): 127-133.

[Benkler 2006] Benkler, Y. (2006) *The wealth of networks: How social production transforms markets and freedom*. New Haven, CT: Yale University Press.

[Bertelli, Bovo, *et al*. 2007] Bertelli, L., Bovo, F., Grespan, L., Galvan, S., and Fiorini, P. (2007). Eddy: an open hardware robot for education. *4th International Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2007)*, Buenos Aires, Argentina.

[Bluedorn, Johnson, *et al.* 1994] Bluedorn, A., Johnson, R., Cartwright, D., and Barringer, B. (1994). The Interface and Convergence of the Strategic Management and Organizational Environment Domains. *Journal of Management*, 20(2): 201-262.

[Boehm 2006] Boehm, B. E.(2006). A View of 20th and 21st Century Software Engineering. *28th International Conference on Software Engineering (ICSE 2006)*, Shanghai, China: 12-29.

[Boehm and Turner 2003] Boehm, B. E. and Turner, R. (2003). *Balancing Agility and Discipline: A Guide to the Perplexed*, Addison-Wesley Professional, New York.

[Bombardieri 2005] Bombardieri, M. (2005). "In computer science a growing gender gap: Women shunning a field once seen as welcoming." *Boston Globe*, December 18, 2005, http://www.boston.com/news/local/articles/2005/12/18/in_computer_science_a_growing_gender_gap/ retrieved August 11, 2006.

[Bonaccorsi and Rossi 2006] Bonaccorsi, A. and Rossi, C. (2006). Comparing Motivations of Individual Programmers and Firms to Take Part in the Open Source Movement. *Knowledge, Technology, & Policy* 18(4): 40-64.

[Bosch 2000] Bosch, J. (2000). *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley Professional.

[Bosch 2009] Bosch, J. (2009). From software product lines to software ecosystems. *Proc. 13th International Software Product Line Conference* (SPLC 2009), San Francisco, CA: 111– 119.

[Bosch and Bosch-Sitjsema 2009] Bosch, J. and Bosch-Sijtsema, P. (2009). From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, 83(1): 67–76.

[Boucharas, Jansen, *et al*. 2009] Boucharas, V., Jansen, S., Brinkkemper, S. (2009). Formalizing software ecosystem modeling. *First International Workshop on Open Component Ecosystems (IWOCE'09)*. Amsterdam, The Netherlands, ACM: 41–50.

[Bradley 2007] Bradley, J-C. (2007). Open Notebook Science Using Blogs and Wikis. *American Chemical Society Symposium on Communicating Chemistry*. Available from *Nature Proceedings,* http://dx.doi.org/10.1038/npre.2007.39.1 accessed June 28, 2010.

[Brown and Booch 2002] Brown, A. W. and Booch, G. (2002). Reusing open-source software and practices: The impact of open-source on commercial vendors. *7th International Conference on Software Reuse: Methods, Techniques, and Tools (ICSR-7),* Austin, Texas, Lecture Notes In Computer Science: 381–428.

[Capiluppi, Morisio, *et al*. 2004] Capiluppi, A., Morisio, M., and Lago, P. (2004). Evolution of Understandability in OSS Projects. *Eighth Euromicro Conference on Software Maintenance and Reengineering (CSMR'04)*, Tampere, Finland.

[Carter 2006] Carter, L. (2006). Why students with an apparent aptitude for computer science don't choose to major in computer science. *37th SIGCSE Technical Symposium on Compute Science Education*, Houston, TX.

[Casadesus-Masanell and Ghemawat 2006] Casadesus-Masanell, R. and Ghemawat, P. (2006). Dynamic Mixed Duopoly: A Model Motivated by Linux vs. Windows. *Management Science,* 52(7): 1072-1084.

[Chadwick 2009] Chadwick, A. (2009) Web 2.0: New Challenges for the Study of EDemocracy in an Era of Informational Exuberance. *I/S: A Journal of Law and Policy for the Information Society,* 5 (1): 9-41.

[Chakravarty, Haruvy, *et al*. 2007] Chakravarty, S., Haruvy, E., and Wu., F. (2007). The link between incentives and product performance in open source development: an empirical investigation. *Global Business and Economics Review 2007,* 9(2/3): 151-169.

[Chan 2004] Chan, A. (2004). Coding Free Software, Coding Free States: Free Software Legislation and the Politics of Code in Peru. *Anthropological Quarterly,* 77(3): 531-545.

[Cheliotis 2009] Cheliotis, G. (2009). From open source to open content: Organization, licensing and decision processes in open cultural production. *Decision Support Systems*, 47(3):229–244.

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

[Citron 2008] Citron, D. K. (2008). Open Code Governance. *University of Chicago Legal Forum, 2008*, U of Maryland Legal Studies Research Paper No. 2008-1: 355-387.

[Clements and Northrop 2001] Clements, P. and Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, New York.

[CMMI 2006] *CMMI for Development (CMMI-DEV)*, Version 1.2, http://www.sei.cmu.edu/library/abstracts/reports/06tr008.cfm accessed June 2010.

[Conway 1968] Conway, M. E. (1968). How do Committees Invent? *Datamation,* 14(5): 28–31.

[Cook, Votta, *et al.* 1998] Cook, J. E., Votta, L. G., and Wolf, A. L. (1998). Cost-Effective Analysis of In-Place Software Processes. *IEEE Transactions on Software Engineering*, 24(8): 650-663.

[COSSHF 2010] *Comparison of Open Source Software Hosting Facilities*, http://en.wikipedia.org/wiki/Comparison_of_free_software_hosting_facilities, accessed 15 July 2010.

[Crowston, et al., 2003] Crowston, K., H. Annabi, & J. Howison. 2003. "Defining Open Source Project Success," In *Proceedings of the 24th Int.l Conf. on Info. Systems*, ICIS, Seattle.

[Crowston and Howison 2006] Crowston, K., and Howison J. (2006). Assessing the Health of Open Source Communities. *Computer*, 39(5): 89 - 91.

[Crowston, Wei, *et al.* 2010] Crowston, K., Wei, K., Howison, J., and Wiggins, A. (2010). Free/libre open source software development: what we know and what we do not know. *ACM Computing Surveys,* (in press).

[Dahlander and Magnusson 2008] Dahlander, L. and Magnusson, M. (2008). How do Firms Make Use of Open Source Communities? *Long Range Planning,* 41(6): 629-649.

[DataOne 2010] *DataOne*, https://dataone.org/ accessed June 28, 2010.

[David 2004] David, P. A. (2004). Understanding the emergence of 'open science' institutions: Functionalist economics in historical context. *Industrial and Corporate Change,* 13(4): 571– 589.

[David and Shapiro 2008] David, P. A. and Shapiro, J. S. (2008). Community-based production of open-source software: What do we know about the developers who participate? *Information Economics and Policy,* 20(4): 364-398.

[David and Spence 2003] David, P. A. and Spence, M. (2003). Towards an Institutional Infrastructure for E-Science: The Scope and Challenge. *Oxford Internet Institute Report No. 2*.

[David, Waterman, *et al*. 2003] David, P. A., Waterman, A., and Arora, S. (2003). FLOSS-US: The Free/Libre/Open Source Software Survey for 2003. *Stanford Institute for Economic Policy Research SIEPR*. Available at http://www.stanford.edu/group/floss-us/. Accessed July 13, 2009.

[Delorey, Knutson, *et al*. 2007] Delorey, D., Knutson, C., Chun, S. (2007). Do Programming Languages Affect Productivity? A Case Study Using Data from Open Source Projects. *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07)*, Minneapolis, MN.

[Deshpande and Riehle 2008] Deshpande, A. and Riehle, D. (2008). The Total Growth of Open Source. *Fourth IFIP International Conference on Open Source Systems (OSS2008)*, Milan, IT, Springer Boston.

[De Souza, Quirk, *et al*. 2007] De Souza, C. R. B., Quirk, S., Trainer, E., and Redmiles, D. F. (2007). Supporting Collaborative Software Development through the Visualization of Socio- Technical Dependencies. *2007 Intern. ACM Conference on Supporting Group Work,* Sanibel Is, FL, ACM Press:147-156.

[Di Penta, German, *et al.* 2010] M. Di Penta, M., German, D. M., Gueheneuc, Y.-G., and Antoniol, G. (2010). An Exploratory Study of the Evolution of Software Licensing. *32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, Cape Town, South Africa, ACM: 145-154.

[Dinkelacker, Garg, *et al.* 2002] Dinkelacker, J., Garg, P. K., Miller, R., and Nelson, D. (2002). Progressive Open Source. *24th International Conference Software Engineering (ICSE 2002)*, Orlando, FL: 177-184.

[Ducheneaut 2005] Ducheneaut, N. (2005). Socialization in a open source software community: a socio-technical analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4): 323-368.

[Dutton 2008] Dutton, W. H. (2008). The Wisdom of Collaborative Network Organizations: Capturing the Value of Networked Individuals. *Prometheus: Critical Studies in Innovation*, 26(3), 211-230. September.

[Ekbia 2009] Ekbia, H. (2009). Digital artifacts as quasi-objects: Qualification, mediation, and materiality. *Journal of American Society for Information Science and Technology*, *60*(12); 2554-2566.

[Ekbia and Gasser 2008] Ekbia, H., and Gasser, L. (2008). Seeking reliability in freedom: The case of F/OSS. In *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing,* K. L. Kraemer & M. Elliott (Eds.), Medford, NJ, Information Today, Inc.: 420-449.

[Elliott and Kreamer 2008] Elliott, M. and Kraemer, K. L. (Eds.) (2008). *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, ASIST Monograph Series, Information Today, Inc.

[Elliott and Scacchi 2003] Elliott, M. and Scacchi, W. (2003). Free Software Developers as an Occupational Community: Resolving Conflicts and Fostering Collaboration. *2003 International ACM SIGGROUP Conference on Supporting Group Work*, Sanibel Island, FL, ACM New York, NY: 21-30.

[Elliott and Scacchi 2008] Elliott, M. and Scacchi, W. (2008). Mobilization of Software Developers: The Free Software Movement. *Information, Technology and People*, 21(1), 4-33.

[English and Schweik 2007a] English, R. and Schweik, C. M. (2007). Identifying Success and Abandonment of Free/Libre and Open Source (FLOSS) Commons: A Preliminary Classification of Sourceforge.net projects. *Upgrade: The European Journal for the Informatics Professional,* Vol. VIII(6).

[English and Schweik 2007b] English, R., and Schweik , C. M. (2007). Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of Sourceforge.net Projects. *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07: ICSE Workshops 2007)*, Minneapolis, MN, IEEE.

[ESA 2007] ESA (European Space Agency). (2007). *European Cooperation for Space Standardisation (ECSS)*, last accessed on June 28, 2010, available at http://www.esa.int/TEC/Software_engineering_and_standardisation/TECMKDUXBQE_2.html

[Everts 2006] Everts, S. (2006). Open Source Science. *Chemical & Engineering News,* 84(30): 34.

[Fang and Neufeld 2009] Fang, Y. and Neufeld, D. (2009). Understanding Sustained Participation in Open Source Software Projects. *Journal of Management Information Systems,* 25(4): 9-50.

[Fernandez-Ramil, Izquierdo-Cortazar, *et al.* 2009] Fernandez-Ramil, J., Izquierdo-Cortazar, D. and Mens, T. (2009). What Does It Take to Develop a Million Lines of Open Source Code? In C. Boldyreff, *et al*. (Eds.) *Open Source Ecosystems: Diverse Communities Interacting*, (IFIP Advances in Information and Communication Technology series), Berlin, Heidelberg, Springer Berlin Heidelberg, Vol 299: 170-184.

[Feller, Finnegan, *et al.* 2008] Feller, J., Finnegan, P., Fitzgerald, B. and Hayes, J. (2008). From Peer Production to Productization: A Study of Socially Enabled Business Exchanges in Open Source Service Networks. *Information Systems Research,* 19(4): 475--494.

[Feiler and Humphrey 1993] Feiler, P. and Humphrey, W. (1993). Software Process Development and Enactment: Concepts and Definitions. *Software Process, 1993. Second International Conference on the Software Process: Continuous Software Process Improvement*, Berlin, Germany: 28-40.

[Fontana, Kuhn, *et al.* 2008] Fontana, R., Kuhn, B. M., Molgen, E., *et al*. (2008). *A Legal Issues Primer for Open Source and Free Software Projects*. Software Freedom Law Center, Version 1.5.1. Retrieved June 28, 2010, 2010, from http://www.softwarefreedom.org/resources/2008/foss-primer.html and http://www.softwarefreedom.org/resources/2008/foss-primer.pdf.

[FOSSology 2010] *FOSSology.* http://fossology.org/about_us (last accessed June 28, 2010).

[Foster 2005] Foster, A. L. (2005). "Student interest in computer science plummets." *The Chronicle of Higher Education*, Washington, DC, 51: A31 – A32.

[Freeman 2007] Freeman, S. (2007). The material and social dynamics of motivation: Contributions to Open Source language technology development. *Science Studies,* 20(2): 55- 77.

[Freeman, Crawford, *et al*. 2005] Freeman, P., Crawford, D., Kim, S., and Munoz, J. (2005). Cyberinfrastructure for science and engineering: Promises and challenges. *Proceedings of the IEEE,* 93(3): 682–69.

[Gabora 1997] Gabora, L. (1997). The Origin and Evolution of Culture and Creativity. *Journal of Memetics - Evolutionary Models of Information Transmission*, 1.

[Gao, Van Antwerp, *et al*. 2007] Gao, Y., Van Antwerp, M., Christley, S., and Madey, G.,

(2007). A Research Collaboratory for Open Source Software Research. *First International Workshop on Emerging Trends in FLOSS Research and Development*, Minneapolis, MN, IEEE Computer Society: 4.

[Garg, Gschwind, *et al*. 2004] Garg, P. J., Gschwind, T., and Inoue, K. (2004). Multi-Project Software Engineering: An Example. *International Workshop on Mining Software Repositories*, Edinburgh, Scotland.

[Gasser , Ripoche, *et al*. 2004] Gasser, L., Ripoche, G. and Sandusky, R. (2004). Research Infrastructure for Empirical Science of F/OSS. *International Workshop on Mining Software Repositories*, Edinburgh, Scotland.

[Gasser and Scacchi 2003] Gasser, L. and Scacchi, W. (2003). Continuous Design of Free/Open Source Software: Workshop Report and Research Agenda. *UCI-UIUC Workshop on Continuous Design of Open Source Software*. (report available http://www.isr.uci.edu/events/ContinuousDesign/Continuous-Design-OSS-report.pdf)

[Gasser and Scacchi 2008] Gasser, L. and Scacchi, W. (2008). Towards a Global Research Infrastructure for Multidisciplinary Study of Free/Open Source Software Development. In *Open Source Development, Community and Quality*, IFIP International Federation for Information Processing, B. Russo, E. Damiani, S. Hissan, B. Lundell, and G. Succi (Eds.), Boston, Springer, Vol. 275: 143-158.

[German and Mockus 2003] German, D. and Mockus, A. (2003). Automating the Measurement of Open Source Projects. *3rd Workshop Open Source Software Engineering,* Portland, OR: 63-68.

[German and Hassan 2009] German, D. and Hassan, A (2009). License Integration Patterns: Addressing license mismatches in component-based development. *IEEE 31st International Conference on Software Engineering,* Vancouver, BC, Canada, IEEE Computer Society: 188-198.

[Gerson 1983] Gerson, E. M. (1983). Scientific work and social worlds. *Science Communication,* 4(3): 357-377.

[Ghosh 2005] Ghosh, R. A. (2005). Understanding free software developers: Findings from the FLOSS study. In *Perspectives on Open Source and Free Software*, J. Feller, B. Fitzgerald, S. A. Hissam, and K. R. Lakhani (ed.), Cambridge, MA: MIT Press: 23-46.

[Ghosh 2006] Ghosh, R. A. (2006). Study on the: Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU. *Official Reports and Studies – 20 November 2006 – EU*

Institutions – Open Source Software, EU Institutions: UNU-MERIT, the Netherlands; Universidad Rey Juan Carlos, Spain; University of Limerick, Ireland; Society for Public Information Spaces, France; Business Innovation Centre of Alto Adige-Südtirol, Italy.

[Ghosh, Glott, *et al*. 2002] Ghosh, R. A., Glott, R., Krieger, B., and Robles, G. (2002). Free/Libre and Open Source Software: Survey and Study. *Workshop on Advancing the Research Agenda on Free / Open Source Software*, Brussels, Belgium.

[Gobeille 2008] Gobeille, R. (2008). The FOSSology project. *2008 International working conference on Mining Software Repositories (MSR '08)*, Leipzig, Germany, ACM: 47-50.

[Godfrey and German 2008] Godfrey, M. W., and German, D. M. (2008). The past, present, and future of software evolution. I*nternational Conference of Software Maintenance and Evolution (ICSM'08): Frontiers of Software Maintenance,* 129-138.

[Gomulkiewicz 2009] Gomulkiewicz, Robert W. (2009). Open Source License Proliferation: Helpful Diversity or Hopeless Confusion? *Washington University Journal of Law & Policy,* 30: 261.

[Gonzalez-Barahona, Robles, *et al*. 2009] Gonzalez-Barahona, J. M., Robles, G., Michlmayr, M., Amor, J. J.,and German, D. M. (2009). Macro-level software evolution: a case study of a large software compilation. *Empirical Software Engineering,* 14(3): 262-285.

[Gould 2002] Gould, S. J. (2002). *The Structure of Evolutionary Theory*, Cambridge, MA, Belknap Press of Harvard University Press.

[Gousios and Spinellis 2009] Gousios, G. and Spinellis, D. (2009). Alitheia core: An extensible software quality monitoring platform. *31st International Conference on Software Engineering,* Vancouver, Canada, IEEE Computer Society: 579-582.

[Gurbani, Garvert, *et al.* 2010] Gurbani, V.K., Garvert, A., and Herbsleb, J.D. (2010). Managing a Corporate Open Source Software Asset, *Communications of the ACM*, 53(2), 155-159.

[Hadoop 2010] *Hadoop* (2010). *Welcome to Hadoop*, http://hadoop.apache.org/. Also see, http://wiki.apache.org/hadoop/ and http://en.wikipedia.org/wiki/Hadoop, Accessed 28 June 2010.

[Haefliger, von Krogh, *et al.* 2008] Haefliger, S., von Krogh, G., and Spaeth, S. (2008). Code Reuse in Open Source Software. *Management Science*, 54(1): 180-193.

[Hahn 2002] Hahn, R. W. (ed.) (2002). *Government policy toward open source software*. Washington, DC: Brookings Institution Press.

[Hahsler and Koch 2005] Hahsler, M. and Koch, S. (2005). Discussion of a Large-Scale Open Source Data Collection Methodology. *38th Annual Hawaii International Conference on System Sciences - Volume 07,* IEEE Computer Society: 197-202.

[Hann, Roberts, *et al.* 2002] Hann, I.-H., Roberts, J., Slaughter, S. and Fielding, R. (2002). Economic incentives for participating in open source software projects. *Twenty-Third International Conference on Information Systems (ICIS) ICIS 2002*, Barcelona, Spain: 365- 372.

[Hann, Roberts, *et al.* 2005] Hann, I.-H., Roberts, J. and Slaughter, S. A. (2005). Why developers participate in open source software projects: An empirical investigation. *Twenty-Fifth International Conference on Information Systems*: 821-830.

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

[Harrison 2001] Harrison, W. (2001). Editorial: Open Source and Empirical Software Engineering. *Empirical Software Engineering*, 6(3), 193-194.

[Hars and Ou 2008] Hars, A. and Ou, S. S. (2002). Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce* 6(3): 25-39.

[Hauge, Ayala, et al. 2010] Hauge, O., Ayala, C. and Conradi, R. (2010). Adoption of Open Source Software in Software-Intensive Organizations - A Systematic Literature Review. *Information and Software Technology*, (in press).

[Heller 1998] Heller, Michael. (1998). The tragedy of the anticommons. *Harvard Law Review*, 111. January.

[Henkel 2006] Henkel, J. (2006). Selective revealing in open innovation processes: The case of embedded Linux. *Research Policy,* 35: 953-969.

[Hertel, Neidner, *et al*. 2003] 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.

[Herz, Lucas, *et al*. 2006] Herz, J.C., Lucas, M., Scott, J. (2006). Open Technology

Development, Roadmap Plan. *DoD Advanced Systems & Concepts,* Version 3.1 (Final). http://www.acq.osd.mil.asc, (accessed 24 June 2010).

[Hey and Trefethen 2005] Hey, T and Trefethen, A. E. (2005). Cyberinfrastructure for escience.*Science,* 308(5723):817–821.

[Hissam, Weinstock, *et al*. 2010] Hissam, S., Weinstock, C., Bass, L. (2010). On Open and Collaborative Software Development in the DoD. Proc. 7th *Annual Acquisition Research Symposium*, Monterey, California.

[Hoepman and Jacobs 2007] Hoepman, J. and Jacobs, B. (2007). Increased security through open source. *Communications of the ACM,* 50(1): 79-83.

[Hofman and Riehle 2009] Hofmann, P., and Riehle, D. (2009). Estimating Commit Sizes Efficiently. In C. Boldyreff, *et al.* (Eds.) *Open Source Ecosystems: Diverse Communities Interacting*, (IFIP Advances in Information and Communication Technology series), Berlin, Heidelberg, Springer Berlin Heidelberg, Vol 299: 105-115.

[Howison 2009] Howison, J. (2009). *Alone Together: A socio-technical theory of motivation, coordination and collaboration technologies in organizing for free and open source software development*. PhD Dissertation. School of Information Studies, Syrcause University. (Available at http://james.howison.name/pubs/dissertation.html)

[Howison, Conklin, *et al.* 2006] Howison, J., Conklin, M., and Crowston, K. (2006). FLOSSmole: A collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering (IJITWE)*, 1(3): 17-26.

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

[Hughes, Lang, *et al.* 2007] Hughes, J., Lang, K. R., Clemons, E. K., and Kauffman, R. J. (2007). *A unified interdisciplinary theory of open source culture and entertainment*. Technical Report 1077909, SSRN.

[Hull, Wolstencroft, *et al.* 2006] Hull, D.,Wolstencroft, K., Stevens, R., Goble, C., Pocock, M. R., Li, P. and Oinn, T. (2006). Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(2): W729–732.

[Iivari 2008] Iivari, N. (2008). Empowering the users? A critical textual analysis of the role of users in open source software development. *AI & Society,* 23(4): 511-528.

[Jansen, Beinkkemper, *et al,*. 2009] Jansen, S., Brinkkemper, S., Finkelstein, A. (2009). Business network management as a survival strategy: A tale of two software ecosystems. *First International Workshop on Software Ecosystems (IWSECO-2009),* S. Jansen, S. Brinkkemper, A. Finkelstein, and J. Bosch (ed.), Falls Church, Virginia: 34-48.

[Jansen, Finkelstein, *et al.* 2009] Jansen, S., Finkelstein, A., and Brinkkemper, S. (2009). A sense of community: A research agenda for software ecosystems. *31st International Conference on Software Engineering (ICSE '09), Companion Volume,* Portland, OR: 187–190.

[Jensen and Scacchi 2004] Jensen, C. and Scacchi, W. (2004). Collaboration, Leadership, Control, and Conflict Negotiation in the NetBeans.org Community. *Proc. Fourth Workshop on Open Source Software Engineering ICSE04-OSSE04*, Edinburgh, Scotland: 48-52.

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

[Jensen and Scacchi 2007] Jensen, C. and Scacchi, W. (2007). Role migration and advancement processes in OSSD projects: A comparative case study. *29th International Conference on Software Engineering*, Minneapolis, MN, ACM: 364-374.

[Jensen and Scacchi 2010] Jensen, C. and Scacchi, W. (2010). Governance in Open Source Software Development Projects: A Multi-Level Comparative Case Study. *Proc. 6th International Conference Open Source Systems,* Notre Dame, IN: 130-142.

[Kaufeler, Jones, *et al.* 2001] Kaufeler, J.-F., Jones, M., and Karl, H.-U. (2001). Promoting ESA Software as a European Product: The SCOS-2000 Example. *ESA Bulletin,* 108: 72-772.

[Kawaguchi, Garg, *et al.* 2003] Kawaguchi, S., Garg, P.K., Matsushita, M., and Inoue, K. (2003). On Automatic Categorization of Open Source Software. *3rd Workshop on Open Source Software Engineering*, Portland, OR: 63-68.

[Kelty 2001] Kelty, C. (2001). Free Software/Free Science. *First Monday,* 6(12).

[Kepler 2010] *The Kepler Project*. https://kepler-project.org/ (last accessed June 28, 2010).

[Klawe, Whitney, *et al.* 2009] Klawe, M. M., Whitney, T., and, Simard, C. (2009). Women in computing - take 2. *Communications of the ACM*, 52(2): 68-76.

[Kling and Scacchi 1982] Kling, R. and Scacchi, W. (1982). The Web of Computing: Computing Technology as Social Action. in *Advances in Computers*, M. Yovits (Ed.), Academic Press, New York, 21: 3-75.

[Koch 2005] Koch, S. (2005). Evolution of Open Source Software Systems—A Large-Scale Investigation. *1st International Conference on Open Source Systems*, Genoa, Italy.

[Kolata 2010] Kolata, G. (2010). Sharing of data leads to progress on Alzheimer's. *New York Times*, 8/13/2010. http://www.nytimes.com/2010/08/13/health/research/13alzheimer.html?_r=1&emc=eta1 (accessed August 16, 2008).

[Krein, MacLean, et al. 2009] Krein, J. L., MacLean, A. C., Delorey, D. P., Eggett, D. L., and Knutson, C. D. (2009). Language Entropy: A Metric for Characterization of Author Programming Language Distribution. *Fourth International Workshop on Public Data about Software Development (WoPDaSD '09),* Skovde, Sweden.

[Kuehnel 2008] Kuehnel, A.-K. (2008). Microsoft, open source and the software ecosystem: of predators and prey—the leopard can change its spots. *Information & Communication Technology Law*, 17(2): 107–124.

[Lakhani, Jeppesen, et al. 2007] Lakhani K.R., Jeppesen, L., Lohse, P., Panetta, J. (2007). *The value of openness in scientific problem solving*. Harvard Business School Working Paper, No. 07–050, Harvard Bus. School, Cambridge, MA

[Lakahani and Wolf 2005] 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 *Perspectives on free and open source software,* Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani (eds.), Cambridge, MA: MIT Press 3-22.

[Lang, Shang, *et al.* 2007] Lang, K. R., Shang, D., and Zicklin, R. (2007). Designing markets for open source production of digital culture goods. *ICEC '07: Proceedings of the ninth international conference on Electronic commerce*, New York, NY, USA. ACM: 283–292.

[Larsen and Klischewski 2004]Larsen, M. H. and Klischewski, R. (2004). Process Ownership Challenges in IT-Enabled Transformation of Interorganizational Business Processes. *Proc. 37th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii.

[Lee 2006] Lee, J.-A. (2006). New Perspectives on Public Goods Production: Policy Implications of Open Source Software. *Vanderbilt Journal of Entertainment and Technology Law*, Vol. 9(1).

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

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

[Lerner and Tirole 2002] Lerner, J., and Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics,* 50 (2): 197–234.

[Lerner and Tirole 2005a] Lerner, J., and Tirole., J. (2005). The economics of technology sharing: Open source and beyond. *Journal of Economic Perspectives,* 19(2): 99-120.

[Lerner and Tirole 2005b] Lerner, J., and Tirole, J. (2005). The Scope of Open Source Licensing. *Journal of Law Economics & Organization,* 21(1): 20-56.

[Lessig 2006] Lessig, L. (2006). *Code: And Other Laws of Cyberspace, Version 2.0.* Basic Books.

[Lewis, 2010] Lewis, J.A. *Government Open Source Policies*. http://csis.org/publication/government-open-source-policies. (Last accessed Sept 3, 2010).

[Lin 2005] Lin, Y. (2005). The future of sociology of FLOSS. *First Monday*, (Special Issue #2: Open Source).

[Long and Siau 2007] Long, Y. and Siau, K. (2007). Social Network Structures in Open Source Software Development Teams. *Journal of Database Management,* 18(2): 25-40.

[MacCormack, Baldwin, *et al*. 2010] MacCormack, A., Baldwin, C., and Rusnak, J. (2010). *The Architecture of Complex Systems: Do Core-Periphery Structures Dominate?* MIT Sloan School of Management Working Paper, 4770-10.

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

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

[Madhavji, Ramil, *et al.* 2006] Madhavji, N.H., Ramil, J.F., and Perry, D. (eds.) (2006). *Software Evolution and Feedback: Theory and Practice*, John Wiley and Sons Inc, New York.

[Markus 2007] Markus, M. L. (2007). The governance of free/open source software projects: Monolithic, multidimensional, or configurational? *Journal of Management and Governance* 11(2): 151-163.

[Messerschmitt and Szyperski 2003] Messerschmitt, D. G. and Szyperski, C. (2003). *Software Ecosystem: Understanding an Indispensable Technology and Industry*. MIT Press, Cambridge, MA.

[Michlmayr and Hill 2003] Michlmayr, M. and Hill, B. M. (2003). Quality and the Reliance on Individuals in Free Software Projects. *3rd Workshop on Open Source Software Engineering: Taking Stock of the Bazaar*, Portland, OR.

[MITRE 2003] MITRE (2003). *Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense*, Version 1.2.04, January 2, 2003. http://cio-nii.defense.gov/sites/oss/2003survey/dodfoss_pdf.pdf (last accessed September 2010).

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

[Morelli, Tucker, *et al*. 2009] Morelli, R.A., Tucker, A., Danner, N., de Lanerolle, T., Ellis, H.J.C., Izmirli, O., Krizanc, D., and Parker, G. (2009). Revitalizing computing education through free and open source software for humanity. *Communications of the ACM*, 52(8): 67- 75.

[Mylyn 2010] Mylyn Eclipse Plugin Project. http://www.eclipse.org/mylyn/ (last accessed June 28, 2010).

[NEES 2010] NEES, *Network for Earthquake Engineering Simulation* https://www.nees.org (last accessed June 28, 2010).

[Niederman 2004] Niederman, F. (2004). IT Employment Prospects in 2004: A Mixed Bag. *Computer*, 37(1): 69-77.

[Nelson and Winter 1985] Nelson, R.R. and Winter, S.G. (1985). *An Evolutionary Theory of Economic Change*, Belknap Press, Cambridge, MA.

[NEON, 2010] *The National Ecological Observatory Network (NEON)*. http://www.neoninc.org (last accessed June 28, 2010).

[Noll 2008] Noll, J. (2008). Requirements Acquisition in Open Source Development: Firefox 2.0. *Open Source Development Communities and Quality*, IFIP International Federation for Information Processing, B. Russo, E. Damani, S. Hissam, B. Lundell, and G. Succi (ed.), Boston, Springer, Vol. 275: 69-79.

[Noll and Scacchi 1991] Noll, J. and Scacchi, W.(1991). Integrating Diverse Information Repositories: A Distributed Hypertext Approach. *Computer*, 24(12): 38-45.

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

[Noll and Scacchi 2001] Noll, J. and Scacchi, W. (2001). Specifying process-oriented hypertext for organizational computing. *Journal Network and Computer Applications*, 24(1): 39-61.

[Northrop, Feiler, *et al.* 2006] Northrop, L., Feiler, P., Gabriel, R. P., Goodenough, J., Linger, R., Longstaff, T., Kazman, R., Klein, M., Schmidt, D. Sullivan, K., and Wallnau, K. (2006). *Ultra-Large-Scale Systems: The Software Challenge of the Future.* Software Engineering Institute, Carnegie Mellon University.

[NSF 2007] National Science Foundation - NSF (2007). *Cyberinfrastructure Vision for 21st Century Discovery.* National Science Foundation, Cyberinfrastructure Council: Arlington, VA. URL http://purl.access.gpo.gov/GPO/LPS80410

[NSF 2010] National Science Foundation - NSF (2010). NSF, *Program Solicitation, Broadening Participation in Computing (BPC)*, Retrieved from http://www.nsf.gov/pubs/2009/nsf09534/nsf09534.html, March 1, 2010.

[Ogawa, Ma, *et al.* 2007] Ogawa, M., Ma, K.-L., Devanbu, P., Bird, C., and Gourley, A. (2007). Visualizing social interaction in open source software projects. *2007 Asia-Pacific Symposium on Visualisation (APVIS'07)*, Sydney, Australia, IEEE Computer Society: 25-32.

[Ogawa and Ma 2008] Ogawa, M. and Ma, K.-L. (2008). StarGate: A Unified, Interactive Visualization of Software Projects. *IEEE PacificVis Conference 2008,* Kyoto, Japan: 191-198.

[O'Neil 2009]O'Neil, M. (2009). *Cyberchiefs: Autonomy and authority in online tribes*. New York, NY: Pluto Press.

[O'Mahony 2007] O'Mahony, S. (2007). The governance of open source initiatives: What does it mean to be community managed? *Journal of Management and Governance,* 11(2): 139- 150.

[O'Mahony and Ferraro 2007] O'Mahony, S., and Ferraro., F. (2007). The emergence of governance in an open source community. *Academy of Management Journal,* 50 (5): 1079-1106.

[Open Design 2010] *Open Design* - http://www.ronen-kadushin.com/Open_Design.asp (last accessed June 28, 2010).

[Open Design Club 2010] *Open Design Club* http://www.opendesignclub.com/ (last accessed June 28, 2010).

[Open Design of Circuits 2010] *Open Design of Circuits* http://www.opencollector.org/history/OpenDesignCircuits/reinoud_index.html (last accessed June 28, 2010).

[Open Hardware Products 2010] *Open Hardware Products* https://fossbazaar.org/content/lca-devbiz-january-2010,(last accessed June 28, 2010).

[OpenSourceCinema.org 2010] *OpenSourceCinema*, Open video production http://www.opensourcecinema.org/ (last accessed June 28, 2010).

[Orman 2008] Orman, W. H. (2008). Giving It Away for Free? The Nature of Job-Market Signaling by Open-Source Software Developers. *The B.E. Journal of Economic Analysis & Policy,* 8(1): Article 12.

[Ossher, Bajracharya, *et al.* 2009] Ossher, J., Bajracharya, S., Linstead, E., Baldi, P., and Lopes. C. (2009). SourcererDB: An Aggregated Repository of Statically Analyzed and Cross- Linked Open Source Java Projects. *6th IEEE Working Conference on Mining Software Repositories (MSR 2009)*, Vancouver, Canada.

[Ovaska, Rossi, et al., 2003] Ovaska, P., Rossi, M. and Marttiin, P. (2003). Architecture as a Coordination Tool in Multi-Site Software Development, *Software Process--Improvement and Practice*, 8(3), 233-247.

[Parastatidis, Viegas, *et al.* 2009] Parastatidis S., Viegas, E., and Hey, T. (2009) Viewpoint: A 'smart' cyberinfrastructure for research. *Communications of the ACM,* 52(12): 33–37.

[Patterson 2006] Patterson, D. A. (2006). Computer science education in the 21st century. *Communications of the ACM*, 49(3): 27–30.

[Paulson, Succi, *et al.* 2004], Paulson, J.W., Succi, C., and Eberlein, A. (2004). An Empirical Study of Open and Close Source Software Products. *IEEE Transactions on Software Engineering*, 30(4), 246-256.

[Pawlowski, Robey, *et al.* 2000] Pawlowski, S. Robey, D., Raven, A. (2000). Supporting shared information systems: boundary objects, communities, and brokering. *Twenty First International Conference on Information Systems*, Brisbane, Queensland, Australia.

[Peccia 2007] Peccia, N. (2007). ESA open-source software supports Germany's TerraSAR-X. ESA, (March 28, 2007), last retrieved June 28, 2010, from http://www.esa.int/esaCP/SEMDV1T4LZE_index_2.html.

[Perens 2005] Perens, B. (2005). The Emerging Economic Paradigm of Open Source. *First Monday*, 10(2).

[Peters 2009] Peters, M. A. (2009). Open Education and the Open Science Economy. *Yearbook of the National Society for the Study of Education*, 108(2), 203-225.

[Raasch, Herstatt, *et al*. 2009] Raasch, C., Herstatt, C. and Balka, K. (2009). On the open design of tangible goods. *R&D Management,* 39(4): 382-393.

[Reding 2007] Reding, V. (2007). *Speech at Truffle 100 event: Towards a European Software Strategy*, European Commission on Information Society and Media, Brussels, http://ec.europa.eu/commission_barroso/reding/docs/speeches/brussels_20071119.pdf ,19 November 2007.

[Rhoten and Powell. 2007] Rhoten, D. and Powell, W. W. (2007). The Frontiers of Intellectual Property: Expanded Protection versus New Models of Open Science. *Annual Review of Law and Social Science*, 3, 345-373.

[Riehle, Ellenberger, *et al.* 2009] Riehle, D., Ellenberger, J., Menahem, T., Mikhailovski, B., Natchetoi, Y., Naveh, B., and Odenwald, T. (2009). Open Collaboration within Corporations Using Software Forges. *IEEE Software*, 26(2): 52-58.

[Ripoche and Gasser 2003] Ripoche, G. and Gasser, L. (2003). Scalable automatic extraction of process models for understanding F/OSS bug repair. *Intern.Conf. Software & Systems Engineering and their Applications (CSSEA'03)*, Paris, France.

[Ripoche and Sameonnet 2006] Ripoche, G., and Sansonnet, J-P. (2006). Experiences in Automating the Analysis of Linguistic Interactions for the Study of Distributed Collectives. *Computer Supported Cooperative Work (CSCW)*, 15(2-3): 149-183.

[Roberts, Hann, *et al.* 2006] Roberts, J., Hann, I.-H., and Slaughter, S. A. (2006). Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science,* 52(7): 984- 999.

[Robles and Gonzalez-Barahona 2006] Robles, G. and Gonzalez-Baharona, J. M. (2006). Contributor Turnover in Libre Software Projects. In *Open Source Systems*, E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto and G. Succi (ed.), Springer Boston, 203: 273- 286.

[Robles, Gonzalez-Barahona, *et al.* 2003] Robles, G., Gonzalez-Barahona, J. M., Centeno-Gonzalez, J., Matellan-Olivera, V., and Rodero-Merino, L. (2003) Studying the evolution of libre software projects using publicly available data. *3rd Workshop on OSS Engineering*, Portland, OR: 111-115.

[Robles, Gonzalez-Barahona, *et al.* 2004] Robles, G., Gonzalez-Barahona, J.M., Ghosh, R., and Carlos, J. (2004) GluTheos: Automating the Retrieval and Analysis of Data from Publicly Available Software Repositories. *International Workshop on Mining Software Repositories*, Edinburgh, Scotland.

[Robles, Gonzalez-Barahona, *et al.* 2006] Robles, G., Gonzalez-Barahona, J. M., Merelo, J. (2006) Beyond source code: the importance of other artifacts in software development. *J. Systems and Software*, 79(9): 1233-1248.

[Robles, Koch, *et al*. 2004] Robles, G., Koch, S., Gonzalez-Barahona, J. M., and Carlos, J. (2004). Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. *2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, Edinburgh, Scotland: 51-55.

[Rosen 2004] Rosen, L. (2004). *Open Source Licensing: Software Freedom and Intellectual Property Law.* Prentice Hall. (Available online at http://www.rosenlaw.com/oslbook.htm)

[Rosenberg 2008] Rosenberg, S. (2008). *Dreaming in Code: Two Dozen Programmers, Three Years, 4732 Bugs, and One Quest for Transcendent Software*, Three Rivers Press.

[Rossi, Russo, *et al.* 2010] Rossi, B., Russo, B., and Succi, G. (2010). Download patterns and releases in open source software projects: A perfect symbiosis? *Open Source Software: New Horizons*, Springer Boston, 319: 252-267.

[Rossi 2006] Rossi, M. A. (2006). Decoding the Free/Open Source (F/OSS) Software Puzzle: A survey of theoretical and empirical contributions. Chapter 2 in *The Economics of Open Source Software Development*. J. Bitzer and P. J. H. Schroder. Amsterdam, Elsevier Press. (2006): 15-56.

[Rowe 2009] Rowe, D. (2009). Open Hardware Business Models. *2009 linux.conf.au Business Development mini conf*, (video of presentation). https://fossbazaar.org/content/david-rowe-open-hardware-business-models (last accessed June 28, 2010).

[Rusovan, Lawford, *et al.* 2005] Rusovan, S., Lawford, M., and Parnas, D. L. (2005). Open Source Software Development: Future or Fad? In J. Feller, B. Fitzgerald, S. A. Hissam, and K. R. Lakhani (eds.),*Perspectives on Free/Open Source Software Development*, J Cambridge, MA: MIT Press: 107-121.

[Sack, Detienne, et al. 2006] Sack, W., Detienne, F., Ducheneaut, Burkhardt, Mahendran, D., and Barcellini, F. (2006). A Methodological Framework for Socio-Cognitive Analyses of Collaborative Design of Open Source Software, *Computer Supported Cooperative Work*, 15(2/3): 229-250.

[Sapp 2009] Sapp, J. (2009). *The New Foundations of Evolution: On the Tree of Life*. Oxford University Press.

[Saviotti and Mani 1995] Saviotti, P. P., and Mani, G. S. (1995). Competition, Variety and Technological Evolution: A Replicator Dynamics Model. *Journal of Evolutionary Economics*, 5(4): 369-392.

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

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

[Scacchi 2004] Scacchi, W. (2004). Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 21(1): 59-66.

[Scacchi 2006] Scacchi, W. (2006). Understanding Open-Source Software Evolution. In *Software Evolution and Feedback: Theory and Practice*, N.H. Madhavji, J.F. Ramil, and D. Perry (eds.), John Wiley and Sons Inc, New York: 181-206.

[Scacchi 2007] Scacchi, W. (2007). Free/Open Source Software Development: Recent Research Results and Emerging Opportunities. *Proc. 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. (ESEC/FSE 2007), Dubrovnik, Croatia, ACM Press, 459–468.

[Scacchi 2008] Scacchi, W. (2008). Emerging Patterns of Intersection and Segmentation when Computerization Movements Interact, in M.S. Elliott and K.L. Kraemer (Eds.), *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, ASIST Monograph Series, Information Today, Inc. 381-404.

[Scacchi 2009] Scacchi, W. (2009). Understanding Requirements for Open Source Software. In *Design Requirements Engineering: A Ten-Year Perspective*, K. Lyytinen, *et al.* (Eds.), Lecture Notes in Business Information Processing, Springer Verlag, Berlin, 14: 467- 494.

[Scacchi 2010a] Scacchi, W. (2010). Collaboration Practices and Affiordanced in Free/Open Source Software Development, in I. Mistrík, J. Grundy, A. van der Hoek, and J. Whitehead, (Eds.), *Collaborative Software Engineering*, Springer, New York, 307-328.

[Scacchi 2010b] Scacchi, W. (2010). Computer Game Mods, Modders, Modding, and the Mod Scene. *First Monday*, 15(5).

[Scacchi and Alspaugh 2008] Scacchi, W. and Alspaugh, T. (2008). Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense. *5th Annual Acquisition Research Symposium*, Naval Postgraduate School, Monterey, CA, NPS-AM-08- 036, Vol. 1: 230-244.

[Scacchi, Feller, *et al.* 2006] Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K. (2006). Understanding Free/Open Source Software Development Processes. *Software Process: Improvement and Practice,* 11(2): 95-105.

[Scacchi, Jensen, *et al.* 2006] Scacchi, W., Jensen, C., Noll, J., and Elliott, M. (2006). Multi- Modal Modeling, Analysis and Validation of Open Source Software Development

Processes. *International Journal on Information Technology and Web Engineering*, 1(3): 49-63.

[Scacchi and Mi 1997] Scacchi, W. and Mi, P. (1997). Process Life Cycle Engineering: A Knowledge-Based Approach and Environment. *Intelligent Systems in Accounting, Finance, and Management*, 6(2): 83-107.

[Schmerken 2009] Schmerken, I. (2009). Wall Street Opens Doors to Open Source Technologies. *Wall Street & Technology*, May 11, 2009. (Available at http://www.wallstreetandtech.com/it%20infrastructure/showArticle.jhtml?articleID=217400216, last accessed 28 June 2010).

[Schryen and Kadura 2009] Schryen, G. and Kadura, R. (2009). Open source vs. closed source software: towards measuring security. *2009 ACM Symposium on Applied Computing SAC '09*, Honolulu, Hawaii, ACM, New York, NY: 2016-2023.

[Schweik 2005] Schweik, C. M. (2005). An institutional analysis approach to studying libre software Commons. *Upgrade: The European Journal for the Informatics Professional,* vol. VI(3): 17-27.

[Schweik and English 2007] Schweik, C. M., and English, R. (2007). Tragedy of the FOSS Commons? Investigating the institutional designs of free/libre and open source software projects. *First Monday,* 12(2).

[Schweik, English, *et al.* 2010]. Schweik, C.M., English, R., Paienjton, Q. and Haire, S. (2010). Success and Abandonment in Open Source Commons: Selected Findings from an Empirical Study of Sourceforge.net Projects *Proceedings of the Sixth International Conference on Open Source Systems (OSS 2010) Workshops*: 91-101.

[Schweik and Kitsing, 2010] Schweik, C.M. and Kitsing, M. "Applying Elinor Ostrom's Rule Classification Framework to the Analysis of Open Source Software Commons" *Transnational Corporations Review* 2.1 (2010): 13-26.

[Seaman and Basili 1998] Seaman, C.B. and Basili, V. (1998). Communication and Organization: An Empirical Study of Discussion in Inspection Meetings. *IEEE Transactions Software Engineering*, 24(7): 559-572.

[Sen, Subramanian, et al. 2008] Sen, R., Subramaniam, C., and Nelson, M. (2008). Determinants of the Choice of Open Source Software Licenses, *J. Management Information Systems*, 25(3), 207-240, Winter.

[Shah 2006] Shah, S. K. (2006). Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science,* 52(7): 1000-1014.

[Simal 2010] Simal - A Project Registry Framework. Retrieved June 28, 2010, from http://simal.oss-watch.ac.uk/.

[Simpson 2003] Simpson, H. R. (2003). Protocols for process interaction. *IEE Proceedings - Computers and Digital Techniques,* 150(3): 157-182.

[Snow, Fjeldstad, *et al*. 2010] Snow, C. C., Fjeldstad, O. D., Lettl, C., and Miles, R. E. (2010). Organizing Continuous Product Development and Commercialization: The Collaborative Community of Firms Model. *Journal of Product Innovation Management*, Winter 2010.

[Sommerville 2006] Sommerville, I. (2006). *Software Engineering, 8th. Edition.* Addison-Wesley, New York.

[SourceForge 2010] *Sourceforge.net*. Available online at www.sourceforge.net , last accessed 24 June 2010.

[Sowe, Stamelos, *et al.* 2006] Sowe, S., Stamelos, I., Angelis, L. (2006). Identifying knowledge brokers that yield software engineering knowledge. *Information and Software Technology*, 48(11): 1025-1033.

[Sjoeberg, D.I.K., et al. 2005] Sjoeberg, D.I.K.; Hannay, J.E.; Hansen, O.; Kampenes, V.B.; Karahasanovic, A.; Liborg, N.-K.; Rekdal, A.C.; (2005) A survey of controlled experiments in software engineering, *IEEE Transactions on Software Engineering,* 31(9): 733- 753.

[Squire and Duvall 2009] Squire, M., and Duvall, S. (2009). Using FLOSS Project Metadata in the Undergraduate Classroom. In *Open Source Ecosystems: Diverse Communities Interacting*, Springer Boston, vol. 299: 330-339.

[Star 1990] Star, S. L. (1990). The Structure of Ill-Structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving. In *Distributed Artificial Intelligence*, L. Gasser and M. N. Huhns, (eds.), Pitman, London, Vol. 2: 37-54.

[Star and Ruhleder 1996] Star, S. L. and Ruhleder, K. (1996). Steps Toward an Ecology of Infrastructure: Design and access for large information spaces. *Information Systems Research*, 7(1): 111-134.

[Starrett 2007] Starrett, E. (2007). Software Acquisition in the Army. *Crosstalk: The Journal of Defense Software Engineering*: 4-8.

[Stewart, Ammeter, et al. 2006] Stewart, K.J., Ammeter, A.P., and Maruping, L.M. (2006). Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects, *Information Systems Research*, 17(2), 126-144, June.

[Strauss 1978] Strauss, A. (1978). A social world perspective. In *Studies in Symbolic Interaction*, Norman Denzin (ed.), Greenwich, Connecticut, JAI Press, Volume 1: 119–128.

[Subramaniam, Sen, et al. 2009] Subramaniam, C., Sen, R., and Nelson, M. L. (2009). Determinants of open source software project success: A longitudinal study. *Decision Support Systems,* 46(2): 576–585.

[Subramanyam and Xia 2008] Subramanyam, R. and Xia, M. (2008). Free/Libre Open Source Software development in developing and developed countries: A conceptual framework with an exploratory study. *Decision Support Systems,* 46(1): 173-186.

[SWEBOK 2004] SWEBOK (2004). *Guide to the Software Engineering Body of Knowledge (SWEBOK),* 2004 edition, IEEE Computer Society. (Available in online form: http://www.computer.org/portal/web/swebok/htmlformat)

[Swedlow and Eliceiri 2009] Swedlow, J. R. and Eliceiri, K. W. (2009). Open source bioimage informatics for cell biology. *Trends in Cell Biology*, 19(11-3 Special Issue - Imaging Cell Biology): 656-660.

[Terry, Kay, *et al.* 2008] Terry, M., Kay, M., Van Vugt, B., Slack, B., and Park, T. (2008). Ingimp: introducing instrumentation to an end-user open source application. *Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems CHI '08*, Florence, Italy, ACM, New York, NY: 607-616.

[Toral, Martinez-Torres, *et al.* 2010] Toral, S.L., Martinez-Torres, M.R., and Barrero, F. (2010). Analysis of virtual communities supporting OSS projects using social network analysis. *Information and Software Technology*, 52: 296-303.

[Tuunanen, Koskinen, et al. 2009] Tuunanen, T., Koskinen, J. and Karkkainen, T. (2009). Automated software license analysis. *Automated Software Engineering*, 16(3-4):455-490.

[UN 2004] United Nations (2004) *Report of the Expert Meeting on Free and Open-Source Software: Policy and Development Implications*. *United Nations Conference on Trade and Development,* Geneva, Switzerland. TD/B/COM.3/EM.21/2. http://www.unctad.org/en/docs/c3em21d2_en.pdf

[Van Antwerp and Madey 2008] Van Antwerp and Greg Madey (2008) Advances in the SourceForge Research Data Archive (SRDA). *The 4th International Conference on Open Source Systems*, *IFIP 2.13 - (WoPDaSD 2008)*, Milan, Italy.

[van Gurp, Prehofer, et al. 2010] van Gurp, J., Prehofer, C., Bosch, J. (2010). Comparing practices for reuse in integration-oriented software product lines and large open source software projects. *Software — Practice & Experience*, 40 (4): 285–312.

[Ven and Mannaert 2008] Ven, K., Mannaert, H. (2008). Challenges and strategies in the use of open source software by independent software vendors. *Information and Software Technology,* 50 (9-10): 991–1002.

[Verma, Jin, *et al.* 2005] Verma, S., Jin, L., and Negi, A. (2005) Open Source Adoption and Use: A Comparative Study Between Groups in the US and India. *Americas Conference on Information Systems (AMCIS 2005)*, Omaha, Nebraska.

[VisTrails 2010] *VisTrails*. http://www.vistrails.org last retrieved June 28, 2010.

[von Hippel 2001] von Hippel, E. (2001). Innovation by user communities: Learning from open-source software. *MIT Sloan Management Review*, 42(4): 82-86.

[von Hippel 2005] von Hippel, E. (2005). *Democratizing Innovation.* MIT Press, Cambridge, MA.

[von Hippel, von Krogh, *et al*. 2003] 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.

[Vortex-Winds 2010] Vortex-Winds, https://www.vortex-winds.org last retrieved June 28, 2010.

[Wasserman and Capra 2007] Wasserman, A. I. and Capra, E. (2007). Evaluating Software Engineering Processes in Commercial and Community Open Source Projects. *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS '07)*, Minneapolis, MN.

[Wasserman, Pal, *et al*. 2006] Wasserman, A. I., Pal, M., and Chan, C. (2006). The Business Readiness Rating Model: an Evaluation Framework for Open Source. *EFOSS – Evaluation Framework for Open Source Software*, Como, Italy.

[Weathersby 2007] Weathersby, J. M. (2007). Open Source Software and the Long Road to Sustainability within the U.S. DoD IT System. *The DoD Software Tech News*, 10(2): 20-23.

[West and Gallagher 2006] West, J. and Gallagher, S. (2006). Patterns of Open Innovation in Open Source Software. Chapter 5 in *Open Innovation: Researching a New Paradigm,* Henry Chesbrough, Wim Vanhaverbeke and Joel West, eds., Oxford: Oxford University Press: 82.

[Wheeler 2007] Wheeler, D.A. (2007) Open Source Software (OSS) in U.S. Government Acquisitions, *DoD Software Tech News*, 10(2), June 2007.

[Wiggins and Crowston 2010] Wiggins, A. and Crowston, K. (2010). Reclassifying Success and Tragedy in FLOSS Projects. *Proceedings of the Sixth International Conference on Open Source Software (OSS 2010),* Notre Dame, IN.

[Wiggins, Howison, *et al*. 2009] Wiggins, A., Howison, A. J., and Crowston, K. (2009). Heartbeat: Measuring Active User Base and Potential User Interest in FLOSS Projects. *Proc. Fifth International Conference on Open Source Software*, Skövde, Sweden, Springer Boston, vol. 299: 94-104.

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

[Wimsatt. and Schank 2004] Wimsatt, W. and Schank, J. C. (2004). Generative Entrenchment, Modularity and Evolvability: When Genic Selection meets the Whole Organism, in *Modularity in Development and Evolution,* G. Schlosser and G Wagner, (Eds.), University of Chicago Press, Chicago: 359-394.

[Wing, Hirsh, *et al.* 2008] Wing, J., Hirsh, H., Kannan, S., and Znati, T. (2008). "*Dear Colleague" Letter: Rethinking Software.* National Science Foundation, CISE. http://www.nsf.gov/cise/news/2008_09_rethink_soft.jsp (Accessed 28 June 2010).

[Weiner 1995] Weiner, J. (1995). *The Beak of the Finch: A Story of Evolution in Our Time*, Vintage Press, New York.

[Weiss, 2005] Weiss, D. 2005. “Measuring Success of Open Source Projects Using Web Search Engines,” *Proceedings of the First International Conference on Open Source Systems*, Genova, 11th-15th July 2005. Marco Scotto and Giancarlo Succi (Eds.), Genoa, 2005, pp. 93-99.

[Wennergren 2009] Wennergren, D.M. (2009). *Clarifying Guidance Regarding Open Source Software (OSS)*, MEMORANDUM Dated October 16 2009, DoD Chief Information Officer, Department of Defense, Washington DC. http://cio-nii.defense.gov/sites/oss/ (last accessed September 2010).

[Xu, Jones, *et al.* 2009] Xu, B., Jones, D. R., and Shao, B. (2009). Volunteers' involvement in online community based software development. *Information & Management,* 46(3): 151-158.

[Ye and Kishida 2003] Ye, Y. and Kishida, K. (2003). Toward an Understanding of the Motivation of Open Source Software Developers. *25th International Conference on Software Engineering (ICSE2003),* Portland, OR, IEEE Computer Society: 419-429.

[Ye, Nakajoki, *et al.* 2005] Ye, Y., Nakajoki, K., Yamamoto, Y., and Kishida, K. (2005). The Co-Evolution of Systems and Communities in Free and Open Source Software Development In *Free/Open Source Software Development,* S. Koch (ed.), IGI Publishing, Hershey, PA: 59-82.