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

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

August 2003

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

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

ABSTRACT

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

Keywords

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

2. Introduction

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

If it is true that F/OSSD is faster, better, and cheaper than SE under certain conditions, then is it possible to see if similar conditions could improve the practices and adapt the principles of SE? Has F/OSSD demonstrated the practical value and success of informal approaches, compared to the formal notation-based approaches advocated by SE scholars? Questions like these cannot be ignored or slighted by mere reference to more than three decades of academic and industrial SE research. Instead, this chapter brings questions like these into the foreground so as to advocate the position that the SE community needs to recognize how, and under what conditions, F/OSSD may represent a faster, better, and cheaper alternative for how to engineer complex software systems. Failure of the SE community to embrace F/OSSD as something different than current SE principles, may relegate the future of SE research to that of an academic curiosity, rather than as an engineering discipline whose capabilities are maximized when operationalized as a complex web of socio-technical development processes and community oriented work practices.

3. Modern software engineering principles and best practices

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

SE is an academic discipline and industrial practice that seeks to rationalize the development of complex software system products and services. It first appeared in the late 1960's, and its principles have been identified, captured, and increasingly taught as a subject suitable for late undergraduate or early graduate study [Brooks 1995, Pressman 2001, Schach 2002, Sommerville 2001]. A quick review of the most current edition of the textbooks cited finds the following principles: First, SE is a team endeavor that is focused on the development of large software systems through a software development life cycle. Second, the software life cycle (model) constitutes a framework that stipulates or orders the processes of SE that every software development project should traverse. Third, the focal processes of SE include software requirements engineering, specification and prototyping, design (functional, architectural, modular, or object-oriented), testing (verification and validation), configuration management, maintenance (or evolution), and project management. Fourth, these processes may or should employ formal notations and reasoning schemes for consistency and completeness, though which computer-based tool to use to support such notations and schemes is unclear. Fifth, software quality results from the systematic performance of software life cycle processes that create, reuse, manipulate, or update software artifacts (including formal notations, graphic diagrams, and source code), according to project planning, cost estimation, and management control efforts. Sixth, the level, goal, or threshold of software quality (e.g., end-user satisfaction, number of defects discovered post delivery) that is sought or attained determines the level of software productivity that is achieved, as well as the overall cost of the software development effort. Following these principles often leads to product development and release cycles that are measured in months to years of calendar time and staff effort.

When these principles of modern SE get applied in industrial centers or in government system acquisition programs, a number of lessons learned emerge which are generally recognized as "best practices" for developing software system products or services through SE. A sample of best practices appears in Exhibit 1 as items grouped according to whether they address project integrity through project management, software construction integrity, and software product stability. These practices are drawn from the Web site of the Software Program Managers

Network[1]. These practices draw attention to risk management, project performance metrics, defect tracking, and testing as a continuous ongoing process, within centrally located and hierarchically organized corporate centers for software production, as supplements to the SE principles already identified above. Together, these principles and practices characterize the technical regime[Nelson and Winter 1982] for developing software products and service through SE.

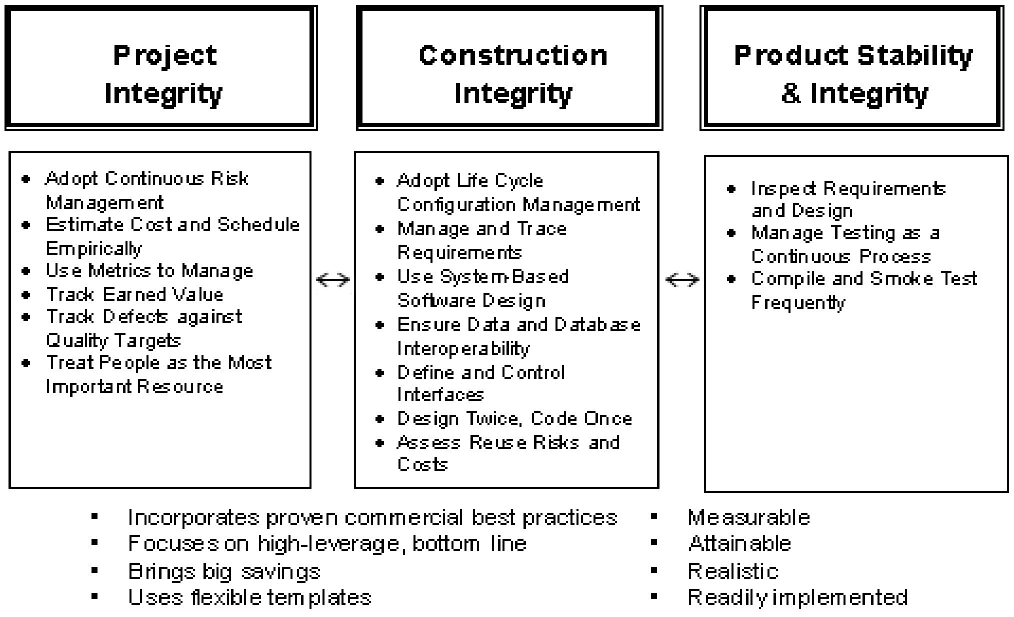

Project Integrity
Adopt Continuous Risk Management
Estimate Cost and Schedule Empirically
Use Metrics to Manage
Track Earned Value
Track Defects against Quality Targets
Treat People as the Most Important Resource
Construction Integrity
Adopt Life Cycle Configuration Management
Manage and Trace Requirements
Use System-Based Software Design
Ensure Data and Database Interoperability
Define and Control Interfaces
Design Twice, Code Once
Assess Reuse Risks and Costs
Product Stability & Integrity
Inspect Requirements and Design
Manage Testing as a Continuous Process
Compile and Smoke Test Frequently
Incorporates proven commercial best practices
Focuses on high-leverage, bottom line
Brings big savings
Uses flexible templates
Measurable
Attainable
Realistic
Readily implemented

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

Finally, with the encouragement and support of the U.S. Department of Defense, industry-wide efforts to improve software product quality have been promoted that entail the external assessment of the "maturity" of a firm's software development capability [Stalk and Hout 1990]that can be observed or measured in terms of the SE principles and best practices that it

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

employs on a routine basis. However, whether or how the assessment of a firm's capability maturity does in fact constitute a reliable indicator or predictor of the quality of the software that is produced is unclear [Beechman 2003, Conradi 2002].

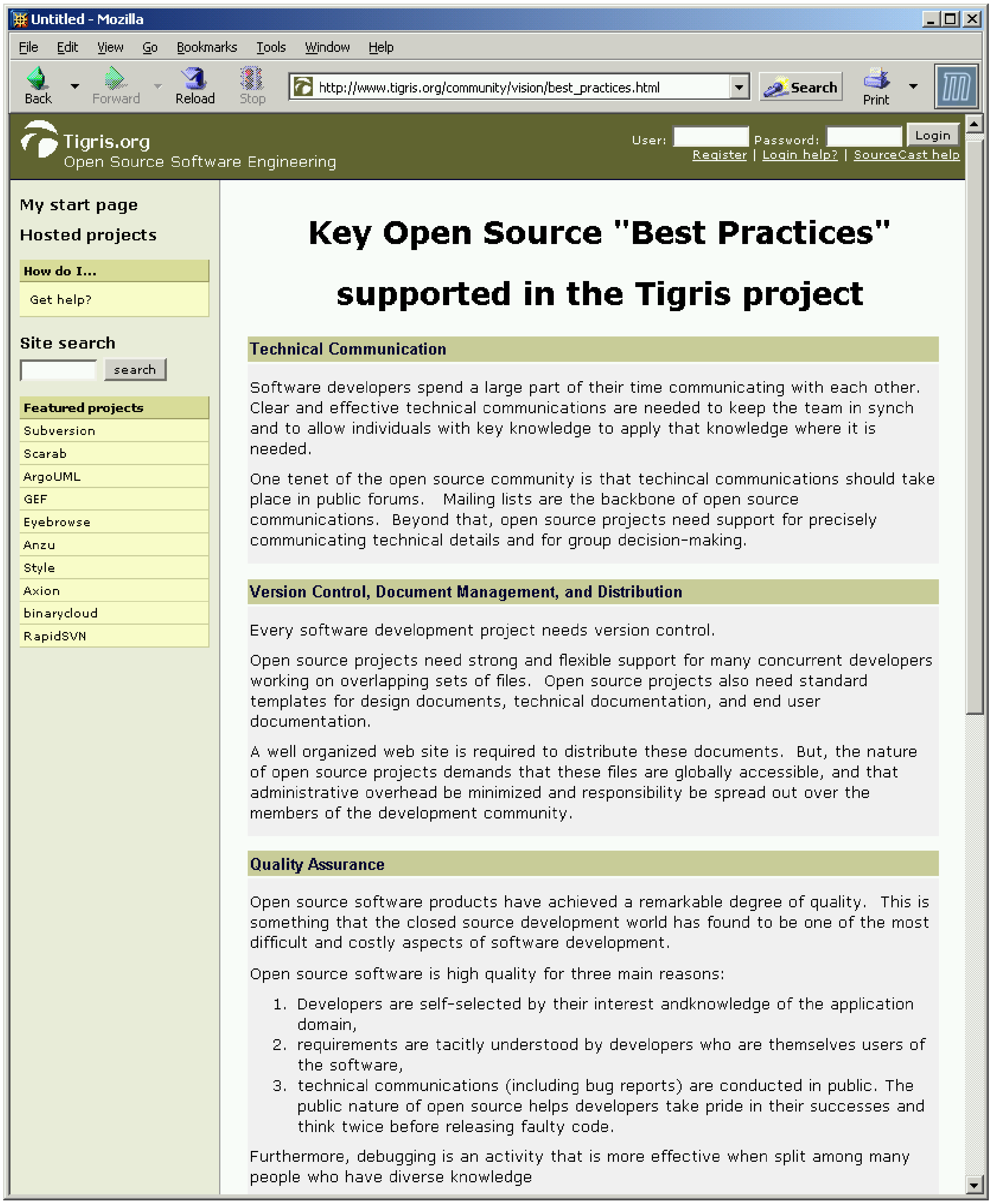
4. Do those who advocate F/OSSD practice modern SE principles?

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

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

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

Third, there are the vendors of OSSD project management environments like SourceForge Enterprise Edition^{TM} from VA Software, SourceCast^{TM} from Collab.Net, and Corporate Source from Zee Source. SFEE, SC, and CS are the products of early commercially oriented OSSD projects that have been evolved and refined into Web-based project management environments for collaborative software development [Augustin 2002]. These environments are not IDEs like NetBeans or Eclipse, though they could be made to interoperate with them. However, these



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

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

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

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

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

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

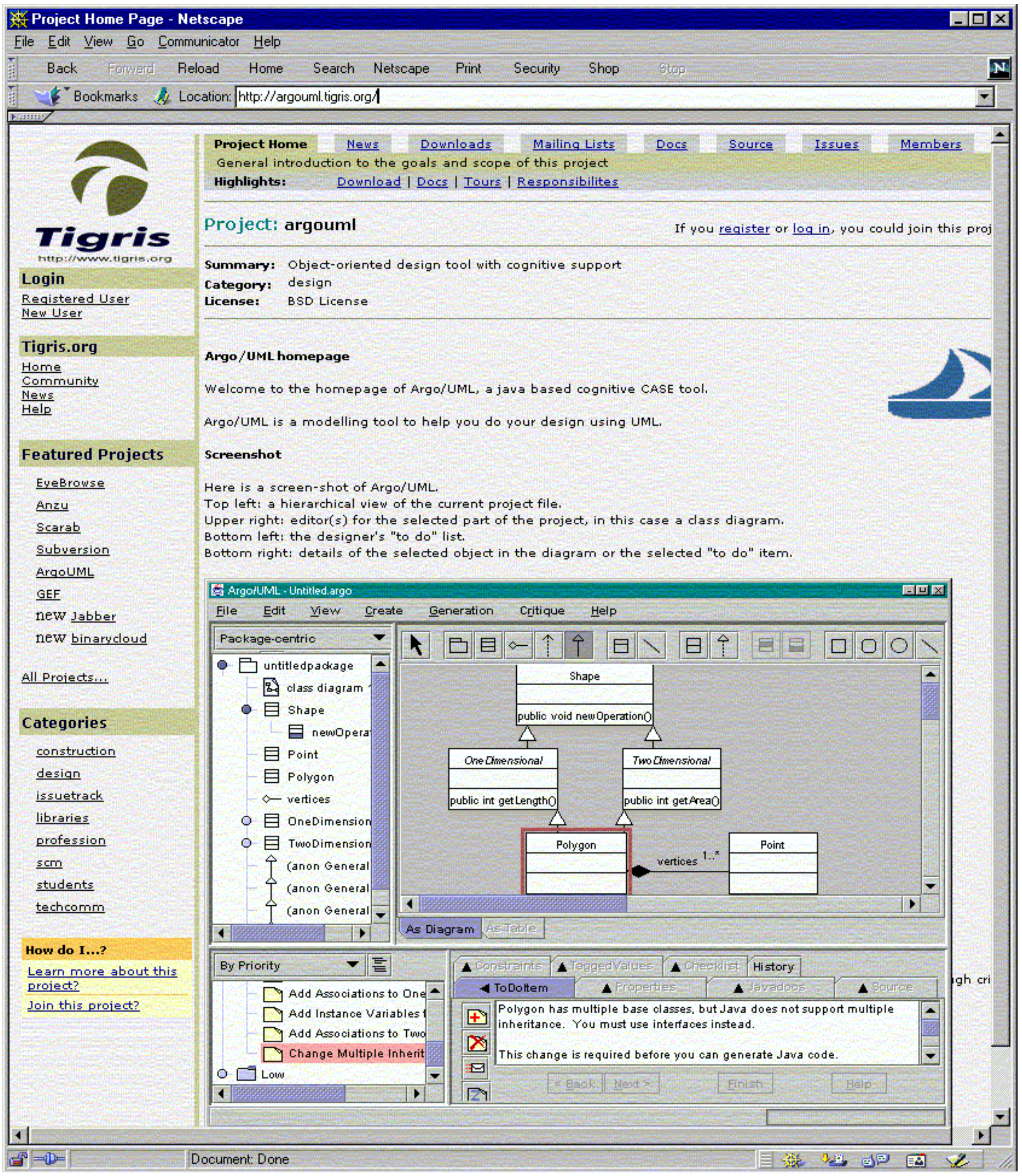
4.1 **Case study:** ***Tigris.org and ArgoUML.tigris.org***

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

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

> *"Tigris.org provides information resources for software engineering professionals and students, and a home for open source software engineering tool projects. We also promote software engineering education and host some undergraduate senior projects."* (http:www.tigris.org, March 2002).

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



General introduction to the goals and scope of this project

Project: argouml

If you register or log in, you could join this proj

Summary:	Object-oriented design tool with cognitive support
Category:	design
License:	BSD License

Argo/UML homepage

Welcome to the homepage of Argo/UML, a java based cognitive CASE tool.

Argo/UML is a modelling tool to help you do your design using UML.

Screenshot

Here is a screen-shot of Argo/UML.
Top left: a hierarchical view of the current project file.
Upper right: editor(s) for the selected part of the project, in this case a class diagram.
Bottom left: the designer's "to do" list.
Bottom right: details of the selected object in the diagram or the selected "to do" item.

Polygon has multiple base classes, but Java does not support multiple inheritance. You must use interfaces instead.

This change is required before you can generate Java code.

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

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

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

The ArgoUML project today includes more than 19,000 registered users and over 150 developers. ArgoUML is thus a large software development project with a significant number of users that is conceived to support SE professionals using modern SE design tools and techniques.

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

> *"Software engineering practices are key to any large development project. Unfortunately, software engineering tools and methods are not widely used today. Even after over 30 years as an engineering profession, most software developers still use few software engineering tools. Some of the reasons are that tools are expensive and hard to learn and use, also many developers have never seen software engineering tools used effectively."* (http://www.argouml.tigris.org, March 2002).

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

> *"The open source software development movement has produced a number of very powerful and useful software development tools, but it has also evolved a software development process that works well under conditions where normal development processes fail. The software engineering field can learn much from the way that successful open source projects gather requirements, make design decisions, achieve quality, and support users. Open source projects are also great for developers to keep their skills current and plug into a growing base of shared experience for everyone in the field."* (http://www.argouml.tigris.org, March 2002).

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

With this modest grounding of exhibits and case study of F/OSSD efforts that in some way address SE topics or concerns, it is possible to examine the overarching question which this chapter addresses. Note that the opening question does not focus on attributes of F/OSS source

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

code programs or other executable implementations (e.g., make files, operating system shell scripts, plug-in modules (like "ActiveX controls"), or intra-application scripting code like JavaScript). Instead focus is directed to attributes of F/OSSD processes, technical regime, management and work practices, and collective community-sustaining actions within projects.

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

What does it mean for one software product development approach to be "faster" than another? Space/time measures like speed come to mind first, but for software development the most common indicators include software productivity (e.g., source lines of code or product feature delivered per developer work time unit [Sommerville 2000]), task duration schedules (process cycle time [Stalk and Hout 1990]), product delivery time (e.g., time to market, time to next product innovation [Kogut and Metiu 2001, von Hippel 2001]), and product defect repair time.

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

Many of the largest and most popular F/OSS systems like the Linux Kernel [Godfrey 2000, Schach 2002b], GNU/Linux distributions [O'Mahony 2003], GNOME user interface [Koch 2002] and others are growing at an *exponential* rate, as is their internal architectural complexity [Schach 2002b]. F/OSS system architectures and functionality can grow in discontinuous jumps as independent F/OSSD projects merge, as their autonomously designed and evolved systems are combined [Nakajoki 2002, Scacchi 2002b]. F/OSSD project teams also continue to grow over time and across releases at an incremental rate, suggesting that adding developers "late" in its development cycle may not slow it down, but may instead increase the size, functionality, and quality of the system. This stands in contrast to the long received wisdom of SE that indicates that adding developers to a project soon before release, delays the release and has an adverse effect on system quality [Brooks 1995]. Thus, there are examples of where F/OSSD projects are producing large software systems whose size and productivity grows at a rate faster than observed for SE projects that focus attention to product development scheduled and management control [cf. Lehman 2002].

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

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

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

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

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

F/OSSD projects are hosted within decentralized communities of peers [Kogut 2001, O'Mahony 2003, Scacchi 2002a, 2002b, Sharman 2002] that can form a virtual enterprise interconnected via logically centralized Web sites and repositories [Noll 1999]. F/OSSD projects rely on their software developers to provide peer reviews of source code and system quality that are synchronized with the complexity of the system release. Major releases undergo more review compared to daily build releases. However, peer review helps create a community of peers, which is one way how social capital may be accumulated (by contributing well regarded review discussions) or lost (by initiating a “flame” discussion that distracts or offends other project contributors and thus discredits the initiator). Community oriented F/OSSD also gives rise to new kinds of requirements for community building, community portals (e.g., SourceForge.net, Freshmeat.net, Savannah.gnu.org, Tigris.org), community software, and community information sharing systems for Web site content management and interlinked communication channels for email, forums, and instant messaging [Scacchi 2002a]. These convivial capabilities and community-oriented mechanisms tend to improve the social welfare of the contributing F/OSSD developers and users. In contrast, most SE projects are targeted for hosting within a centralized corporate setting, where access and visibility may be restricted to local participants under the administrative control of project or business managers. But F/OSS systems also co-evolve with the community of developer-users who create and contribute to them [Nakajoki 2002, O’Mahony 2003, Scacchi 2002a]. However communities, as a form of social organization for software development and for learning about software technology practices, are not mentioned in modern SE principles, practices or textbooks.

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

Finally, it appears that F/OSSD projects rely on *virtual project management* to mobilize, coordinate, control, build, and assure the quality of F/OSSD activities. VPM invites or encourages system contributors to come forward and take a shared, individual responsibility that will serve to benefit the F/OSSD collective of user-developers. VPM requires multiple people to come forward to act in the role of team leader, sub-system manager, or system module owner in a manner that may be short-term or long-term, based on their skill, accomplishments, availability and belief in community development [Fielding 1999]. In contrast, SE projects are predicated on centralized project management regimes where project managers are assigned the administrative authority to plan, manage, and control software development resources, staff, schedule, and budget their projects. F/OSSD projects enable participative management that can arise from a decentralized community of developers, whereas SE assumes delegative management that arises from a centralized corporate structure and resource control framework.

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

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

F/OSSD tools and application systems are inexpensive/free to acquire, comparatively easy to use and learn, and are globally accessed and transferred across the Internet [DiBona 1999, Pavlicek 2000]. In many situations, there are competing alternative implementations of F/OSS tools, so that developers can assess and evaluate which one best meet their need or taste. These tools are both given and received as public goods or gifts [Bergquist 2001]. F/OSS operating environments like the Debian GNU/Linux distribution constitute thousands of utilities, tools and end-user applications, whose overall development costs is estimated in billions of dollars [Gonzalez-Barahona 2003, O'Mahony 2003], are available as F/OSS for immediate download from the Internet. Substantial F/OSSD tool collections [Halloran 2002] are essentially free, therefore have fewer cost barriers to their procurement and adoption, and are more readily transferred within and across communities of developers and users. Commercially available SE tools like Rational Rose^{TM} or Microsoft Visual Studio.Net^{TM} are not free, though they may be available for free trail periods of days or weeks. However, they may have barriers to their adoption in form of perceived higher costs in their acquisition, training, number of user licenses, and product support. Commercially available collaborative software development environments like SFEE, SC and CS, though also not free, are packaged for sale with support services that similarly represent the major cost of their procurement and sustained use.

F/OSS tools and applications are among the most widely used examples that demonstrate the potential for software reuse [Brown 2002]. Successful and widespread reuse saves time and reduces development costs by avoiding redevelopment of previously developed products, components or modules. However, F/OSSD encourages not only software reuse and resource sharing, but in many ways encourages the ongoing evolution of tools and applications through reinvention as a basis for continuous improvement. Faster and better F/OSSD conditions in turn tend to drive down the cost of developing software, at least in terms of schedule and budget resources. SE of course also encourages software reuse through advocacy of component-based system design and use of commercial-off-the-shelf (COTS) components. However, it may be the situation that SE encourages invention over reinvention, and relies primarily on corporate initiatives for software process improvement, but with mixed results [Beechman 2003, Conradi 2002], instead of developer-user motivation for software product improvement [Hars 2002, von Hippel 2001].

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

8. Discussion

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

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

F/OSSD is not a panacea compared to SE, nor is it without its shortcomings. As noted, the vast majority of F/OSSD projects fail to grow beyond 1-2 developers, and subsequently their associated software source code never achieves a critical mass of users, functionality, community discourse informalisms or related resource subsidies. So F/OSSD in general is a risky undertaking, at least in terms of the probability of achieving critical mass, as well as realizing a faster, better, and cheaper way to develop complex software products or services. Accordingly, F/OSSD is not well suited for adoption in hierarchical organizations that develop software products or services through rational management schemes traditional to SE principles and practices [cf. Dinkelacker 2002]. Organizations or firms that develop software products or service that are not wedded to the tradition of SE may on the other hand, find the adoption of F/OSSD practices as a viable alternative, even when developing high-value software products like enterprise resource planning (ERP) systems.[3] Even large software development companies like IBM, SUN, HP, and SAP have all started or sponsored F/OSSD projects (IBM Eclipse, SUN NetBeans, HP Gelato, SAP DB-MySQL) that are separate from, but complementary to, their mainstream software product lines that tend to follow SE practices.

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

Last, those who teach SE should consider how to embrace F/OSSD practices, community information resources and infrastructure as an alternative approach to the development of large software system products or services. These practices build viable lateral social relationships that are cultivated, sustained, and evolved through the ongoing use, sharing, and reuse of F/OSSD informalisms. These informalisms encourage the sharing, study, reinvention, modification and redistribution of development project results. SE education need not be wedded only to the interests of rationally managed and hierarchically organized corporate software development

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

projects. SE educators may also consider where and how to embrace a more free and open global community of students and collaborating software engineers who want to improve their software development productivity and quality, while reducing its costs. Finally, current SE textbooks are in need of revision to accommodate emerging F/OSSD principles and practices.

9. Conclusions

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

F/OSSD is faster than SE when the development life cycle is focused on Internet time product releases that can be produced and delivered on a daily basis. The rapid development and release cycle means that unstable but operational systems are delivered most of the time, while stable systems emerge slowly after extensive community review, iterative and incremental refinement, online discourse about system features and usage experience. Large F/OSS systems may grow in size and platform diversity at a faster rate than systems resulting from SE, and these F/OSS systems are of acceptable quality to users. Independent F/OSSD projects can merge into larger projects which may then reach a critical mass of developers needed to obtain high growth rates in system functionality and quality. However, SE may be able to contribute expertise for how to restructure and reinvent the architecture of large F/OSS systems, so as to continuously improve and mitigate their unwanted complexity. Finally, most developers of F/OSS systems are themselves users of these same systems, thus they can more readily determine system requirements and design features through online discourse. All of these capabilities help make F/OSSD faster than traditional SE principles and practices.

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

F/OSSD is cheaper than SE when the total costs of F/OSS tools and end-users applications is low or free, though their collective development cost enables the flow of social capital among their developers and users. F/OSSD projects are a prime venue for demonstrating the value of software reuse and reinvention, and this speeds development and reduces its costs. Many F/OSS developers appear motivated to give away the products of their collaborative software development work in order to share, examine, learn, reinvent, modify, and redistribute the results of their experiences and practice of F/OSSD in ways that build and sustain a community of like-minded developers.

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

10. References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Lehman, M.M., 2002. Software Evolution, in J. Marciniak (ed.), *Encyclopedia of Software Engineering,* 2nd Edition, John Wiley and Sons Inc., New York, 1507-1513.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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