Is 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, +1-949-824-1715
Wscacchi@uci.edu

ABSTRACT
In this paper, I draw attention to the question of determining how open source software development may represent a significant alternative to modern software engineering techniques for developing large-scale software systems. OSSD often entails shorter time frames, producing higher quality systems, and incurring lower costs than may be realized through developing systems according SE techniques. Understanding why and how this may arise is the focus of this paper.

Keywords
Software Engineering, Open Source Software Development

1. Introduction
The likelihood and circumstances in which open source represents a more effective and efficient approach than software engineering merits serious review. Such conditions may point to the need to critically reflect on how the practice and principles of software engineering needs a serious rethinking and possible reformulation to address and accommodate OSSD, as well as how OSSD differs from current SE principles.

If it is true that OSSD is faster, better, and cheaper than SE, then why bother with SE? Does OSSD address and resolve the "software crisis" that gave rise to SE? Has OSSD demonstrated the practical value and success of informal approaches, compared to the formal notation-based approaches widely advocated SE scholars? Is "humanated" OSSD more productive than automated SE? Answering these questions cannot be ignored or slighted by mere reference to more than three decades of academic and industrial SE research. Instead, this position paper seeks to bring questions like these into the foreground so as to advocate the position that the SE community needs to recognize how, and under what conditions, OSSD may represent a faster, better, and cheaper alternative for how to engineer complex software systems. Failure of the SE community to embrace OSSD as something different than current SE principles, may relegate the future of SE research to that of an academic curiosity, rather than as an engineering discipline whose capabilities are maximized when operationalized as a complex web of socio-technical development processes and community oriented work practices.

2. Do those who advocate "open source software engineering" practice modern software engineering principles?
To help motivate my position, I first present a small case study that examines one experience that tries to show that OSSD and SE are not necessarily the same in means and ends.

2.1 Case study: *Tigris.org*
Let us consider the following case of the Tigris.org OSSD community and some of the OSSD/SE projects affiliated with it. None of the observations that follow are intended to in any way 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 OSSD and SE come together.

This community identifies itself (via the Web site) as being a meeting ground for OSS developers and software engineering 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 software engineering techniques and concepts being applied to support OSSD. For example, OSSD seems to focus attention to source code development and debugging [DiBona 1999, Pavilcek 2000]. Thus modern coding techniques like modularity and the use of program debugging and execution monitoring tools are expected. Beyond this, most SE textbooks draw attention to topics like requirements engineering, software architecture and component design, validating an implementation (i.e., source code) satisfies its requirements, while testing/verifying the implementation is a consistent, complete, traceable, and in some way correct realization of its architecture and component design. Project management and configuration management also receives appropriate attention. So we next look to find examples and instances of these SE practices in Tigris's "featured projects", such as ArgoUML.

ArgoUML "*is a modelling 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). The ArgoUML project today includes more than 19,000 registered

users and over 150 developers[1]. 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 textbooks, and found in use in many SE R&D projects. Using UML, software engineering professionals can create or document Use Cases for software requirements, and of course, 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 we, as software engineering professionals suppose to learn from the ArgoUML experience in SE? 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, if we examine any of the remaining 35 or so other projects affiliated with Tigris.org, it is difficult to find which 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, but not generalizing to all OS-SE efforts, it appears that the objectives and practices of OSSD and SE are different, and may not be closely related. These results also should help researchers investigating OSSD projects recognize the potential risks for making pre-mature generalizations about typifying what OSSD is, or how it works, based on the examination of a single OSSD project, or even a single OSSD community [cf. Scacchi 2001, 2002]. What is true of one OSSD project's artifacts, processes, or practices may not be true of any other OSSD project, without explicit comparison.

With this modest grounding, I now turn to more examine the overarching question which my position addresses. Note that my opening question does not focus on attributes of open source 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 I focus on OSSD processes and practices.

3. How is OSSD *faster* than SE?

OSSD projects like those at Tigris.org enact "Internet time" development practices, much like Microsoft, Netscape, and others [Cusumano 1999, MacCormack 2001]. Internet time efforts emphasize minimizing time to market and delivery of incremental improvements in functionality, instead of complete well-engineered functionality. Internet time development also focuses on collecting feedback from early users as a way to determine which incremental functionality, and which perceived errors in available functionality matters most, as a way to determine emerging system requirements after the fact [Truex 1999]. OSSD projects rely on software *informalisms* [Scacchi 2002] as information resources that can be browsed, crosslinked, and updated on demand. These informalisms are socially lightweight mechanisms for managing, communicating, and coordinating globally dispersed knowledge about who did what, why, and how. These informalisms are easy to learn and use as semi-structured representations that capture software requirements, system design, and design rationale. As OSS developers are themselves end-users of their systems, then software requirements and design take less time to articulate and negotiate, compared to SE projects that must elicit requirements and validate system design with end-users who are generally not SE professionals.

4. How is OSSD *better* than SE?

OSSD projects are iteratively developed, incrementally released, reviewed and refined by software development peers in an ongoing agile manner [cf. Boehm 2001]. These methods ensure acceptable levels of quality, coherence, and security of system-wide software via continuous distributed testing and profiling [Payne 2002, Schmidt 2001]. OSSD efforts are hosted within decentralized communities of peers [Benkler 2001, Kogut 2001, Scacchi 2001, 2002, Sharman 2002] that is interconnected via Web sites and repositories. Community oriented OSSD also gives rise to new kinds of requirements for community building, community software, and community information sharing systems (Web site and interlinked communication channels for email, forums, and chat). In contrast, most SE projects are targeted for hosting within a centralized corporate setting, where access and visibility may be restricted to local participants. OSSD standards [Freericks 2001] that reinforce best practices are apparently easier to access and follow due to their Web-based deployment, and a long history of community oriented participation in developing implementation oriented standards in an open source manner,

---

[1] http://www.tigris.org/community/community_main.html

[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 OSSD projects, but instead to highlight that OSSD and SE practices are different.

compared to the institutionally oriented processes used to develop SE standards.

5. How is OSSD *cheaper* than SE?

OSSD tools are inexpensive/free, comparatively easy to use and learn. These tools are both given and received as public goods or gifts [Bergquist 2001]. Faster and better OSSD conditions in turn tend to drive down the cost of developing software, at least in terms of schedule and budget resources. Most OSSD projects are voluntarily staffed who want to work on the project, who will potentially commit their own time and effort, and who find personal and professional benefit from the OSSD development efforts [Scacchi 2002]. Minimal management or governance forms [Fielding 1999, Sharman 2002] are used to direct OSSD efforts, compared to the more rigidly hierarchical, managed, planned, staffed, controlled, and budgeted project activities typical for SE efforts.

6. How to make SE faster, better, and cheaper via OSSD processes and practices

OSSD projects enact teamwork structures and relatively flat, peer-oriented decentralized community forms [Benkler 2001, Kogut 2001, Scacchi 2001, 2002, Sharman 2002] that reduce/supplant functional organizational forms inherent in traditional SE techniques that increase bureaucratic tendencies. OSSD avoids reliance on formal project management techniques and administrative structures that pervade industrial SE projects. OSSD is generally community oriented and agile [Boehm 2002], rather than customer oriented and formal [Scacchi 2002]. Developers as users mitigate need to spend resources trying to figure out what users want, and whether what is developed and delivered meets user needs [Scacchi 2002]. Thus, the opportunity exists for developing new SE processes, practices, and community forms that are decentralized, peer-oriented, and rely on semi-structured, informal representations of software artifacts. SE community Web sites and community development tools also appear to be candidates for adoption.

7. Conclusions

OSSD appears to be changing the world of software development at a faster, better, and cheaper pace, and with a broader impact and audience, than SE has achieved. Understanding why this is so may be key to advancing the state of the art of both SE and OSSD. Failing to recognize the differences between the two may result in OSSD characterizing more of the leading edge of software system development, while SE characterizes more of the trailing edge. Where do you want to be?

8. Acknowledgements

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

9. References

1. Y. Benkler, The Battle Over the Institutional Ecosystem in the Digital Environment, Communications ACM, 44(2), 84-90, February 2001.
2. M. Bergquist and J. Ljungberg, The power of gifts: organizing social relationships in open source communities, *Info. Systems J.*, 11(4), 305-320, October 2001.
3. B. Boehm, Get Ready for Agile Methods, with Care, *Computer*, 35(1), 64-69, Jan. 2002.
4. M. Cusumano and D.B. Yoffie, Software Development on Internet Time, *Computer*, 32(10), 60-69, October 1999.
5. C. DiBona, S. Ockman and M. Stone, *Open Sources: Voices from the Open Source Revolution*, O'Reilly Press, 1999.
6. J. Feller and B. Fitzgerald, *Understanding Open Source Software Development*, Addison-Wesley, NY, 2002.
7. R.T. Fielding. Shared Leadership in the Apache Project. *Communications ACM*, 42(4), 42-43, 1999.
8. C. Freericks, Open Source Standards on Software Process: A Practical Approach, *IEEE Comm. Mag.,*116-123, April 2001.
9. N. Jørgensen, Putting it all in the trunk: incremental software development in the FreeBSD open source project, *Info. Systems J.*, 11(4), 321-336, October 2001.
10. B. Kogut and A. Metiu, Open Source Software Development and Distributed Innovation, *Oxford Review of Economic Policy*, **17**(2), 248-264, 2001.
11. A. MacCormack, R. Verganti, and M. Iansiti, Developing Products on Internet Time: The Anatomy of a Flexible Development Process, *Mgmt. Science*, 47(1), January 2001.
12. A. Mockus, R.T. Fielding, and J. Herbsleb, A Case Study of Open Source Software Development: The Apache Server, *Proc. 22nd. Intern. Conf. Soft. Eng*, 263-272, 2000.
13. R. Pavlicek, *Embracing Insanity: Open Source Software Development*, SAMS Publishing, Indianapolis, IN, 2000.
14. C. Payne, On the Security of Open Source Software, *Info. Systems J.*, 12(1), 61-78, January 2002.
15. W. Scacchi, Software Development Practices in Open Software Development Communities, *1st. Workshop on Open Source Software Engineering,* Toronto, Ontario, May 2001.
16. W. Scacchi, Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings - Software*, to appear, 2002.
17. D. Schmidt and A. Porter, Leveraging Open-Source Communities to Improve the Quality & Performance of Open-Source Software, *1st. Workshop on Open Source Software Engineering, Toronto, Ontario, May 2001.*
18. S.Sharman, V. Sugurmaran, and B. Rajagopalan, A Framework for Creating Hybrid-Open Source Software Communities, *Info. Systems J.,* 12(1), 7-25, 2002.
19. D. Truex, R. Baskerville, and H. Klein, Growing Systems in an Emergent Organization, *Communications ACM*, 42(8), 117-123, 1999.