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

(Position Paper)

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

April 2001

Overview

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

Understanding open software development practices

Our interest is in understanding the practices and processes of open software development in different communities. We assume there is no *a priori* model or globally accepted framework that defines how open software is or should be developed. So our starting point is to investigate open software practices in different communities to be able to identify what communities members believe are their best practices (e.g., http://www.tigris.org/community/vision/best_practices.html for a description of best practices for open software development advocated in the ArgoUML/Tigris.org project).

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

- *Networked computer game worlds*--first person shooters (e.g., Quake Arena, Unreal Tournament), massively multiplayer online role-playing games (MMORPG, e.g.,

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

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

Comparative case study framework

The software development practices of the four communities we chose to examine can be compared and contrasted in a number of ways. In this regard, we are conducting case studies in each community. Observations and findings from each such case study can be studied, analyzed, and compared:
- As individual cases (e.g., software development practices associated with the ArgoUML software design tool vs. practices associated with the Apache Web server vs. scholarly research papers describing studies of each tool (Mockus, Fielding, and Herbsleb 2000 (Apache), Robbins and Redmiles 2000 (ArgoUML)).
- Multiple cases within a community (e.g., game "mods" (Cleveland 2000) made by users of first person shooters vs. MMORPG vs. others)
- Multiple cases across two communities (e.g., X-ray astrophysics vs. software design are both primarily research oriented communities; games vs. Internet infrastructure are primarily development oriented communities, though these distinctions are not absolute)
- Multiple cases across all communities

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

Comparative case studies are important in that they can serve as foundation for the formalization of our findings and process models as a *process meta-model* (Mi and Scacchi 1996). Such a meta-model can be used to construct a predictive, testable, and incrementally refined *theory* of open software development processes within or across communities or projects. A process meta-model is also used to configure, generate, or instantiate Web-based process modeling, prototyping, and enactment environments that enable modeled processes to be globally deployed and computationally supported (Scacchi and Noll 1997, Noll and Scacchi 1999). This may be of most value to other academic research or commercial development organizations that seek to adopt open software "best practices" or development process that are most suited to their needs and situation (Smarr and Graham 2000). Subsequently, we now turn to highlight the software development processes that have been put into practice within different open software communities.

Open software development processes

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

From our studies to date, we have found five types of software development processes being employed across all four communities. Each process is briefly described in turn, though none should be construed as independent or more important than the others should. Furthermore, it appears that these processes may occur concurrent to one another, rather than strictly or partially ordered within a traditional life cycle model. As before, examples of these processes (i.e., process instances) can be provided for presentation at the Workshop and in the full paper.

Requirements analysis and specification

Software requirements analysis helps identify what problems a software system is suppose to address, while requirements specification identify an initial mapping of problems to system based solutions. In open software development, how does requirements analysis occur, and where and how are requirements specifications described? At this point in the study, we have yet to discover any sort of records of formal requirements elicitation, capture, and analysis activity of the kind suggested by modern software engineering textbooks in any of the four communities under study. Similarly, we have not yet found any online (in the Web sites) or offline (in published technical reports) of documents identified as "requirements specification" documents. What we have found is different.

It appears at this time that open software requirements are manifested as threaded messages or discussions that are captured and/or posted on a Web site for open review, elaboration, refutation, or refinement. Requirements analysis and specification are *implied activities* that routinely emerge as a by-product of community discourse about what their software should or should not do, as well as who will take responsibility for realizing such requirements. Open software system requirements appear in the form of situated discourse within private and public email discussion threads, emergent artifacts (e.g., source code fragments included within a message) and dialectical social actions that negotiate interest, commitment, and accountability (Goguen 1996, Truex, Baskerville, and Klein 1999). More conventionally, requirements analysis and specification do not have first-class status as an assigned or recognized task. Similarly, there are no software engineering tools used to support their capture, negotiation, and "cost" (e.g., level of effort, expertise/skill, and timeliness) estimate, though each of these activities regularly occurs. Nonetheless requirements do exist, though finding or recognizing them demands familiarity and immersion within the community and its discussions. This of course stands in contrast to efforts within the academic software engineering community to develop and demonstrate tools for explicitly capturing requirements, negotiating trade-offs among system requirements and stakeholder interests, and constructive cost estimation or modeling (Boehm, Egyed, *et al.*,1999).

Coordinated version control, system build, and staged incremental release

Software version control tools such as the concurrent versions system, CVS (Fogel 1999--itself an open software system and document base), have been widely adopted for use within open software communities. It is clear, however, that CVS is being used as both a centralized community software coordination mechanism, as well as a venue for mediating control over what software enhancements, extensions, or upgrades will be *checked-in* and made available throughout the decentralized community as part of the publicly released version. Software version control, as part of the software configuration management process, is a recurring situation that requires articulation work (Grinter 1995, Mi and Scacchi 1991). Articulation work is required due to the potential tension between centralized decision-making authority and decentralized work activity when two or more autonomously contributed updates are made which overlap, conflict with one another, or generate unwanted side-effects. Each community or CVS repository administrator must decide what can be checked in, and who will or will not be able to check-in new or modified software source code representations. Sometimes these policies are made explicit through a voting scheme (Fielding 1999), while in others they are left informal, implicit and subject to renegotiations. In either situation, version updates must be coordinated in order for a new system build and release to take place. Subsequently, those developers who want to submit updates to the community's shared repository rely extensively on discussions that are supported using "lean media" such as informally threaded email messages (Yamauchi, Yokozawa, et al., 2000) posted on a Web site, rather than onerous or opaque system configuration rules. Thus, coordinated version control, system build and release is a process that is mediated by the joint use of versioning, system building, and messaging tools.

Maintenance as evolutionary redevelopment, refinement, and redistribution

Software maintenance, as construed to include the addition/subtraction of system functionality, debugging, restructuring, tuning, conversion (e.g., internationalization), and migration (across platforms), is a dominant, recurring process in open software development communities. Perhaps this is not surprising since maintenance is generally viewed as *the* major cost activity associated with a software system across its life cycle. This traditional characterization of software maintenance does not do justice for what can be observed to occur within different open software communities.

Open software systems seem to evolve in a manner similar to fruit flies (*Drosophila melanogaster*--see http://sdb.bio.purdue.edu/fly/aimain/1aahome.htm) through minor genetic mutations that are expressed (articulated and reproduced), recombined, and redistributed across many generations of short life cycles. This is not the same as "genetic programming", which is a synthetic technique for automatically generating semi-random alterations to a predefined software program seed. Instead, what we see is socially constructed mutations (see the Requirements section above) that articulate and adapt what an open software system is suppose to do (Bendifallah and Scacchi 1987). These modifications or updates are then expressed as a tentative new version that may survive redistribution, and subsequently be recombined and re-expressed with other new mutations in producing a new generation version.

Project management

Community software development can take the form of a "pyramid meritocracy" (cf. Kim 1999, Fielding 1999) operating as an Internet-based virtual enterprise (e.g., Fielding, Whitehead, *et al.,* 1998, Noll and Scacchi 1999). A pyramid meritocracy is a hierarchical organizational form that centralizes and concentrates certain kinds of elite authority, trust, and respect for experience and accomplishment within the community. Meritocracy is also ironically is a term that connotes the predilection of an emerging elite who become increasingly self-serving and self-limiting, rather than embracing imaginative, courageous, or generous departures from the status quo (Young 1958). Meritocractic enterprises embrace incremental innovation (e.g., software maintenance work (Bendifallah and Scacchi 1987) and incremental enhancement to an existing software code base) over radical innovation (exploration or adoption of untried or sufficiently different software development methods, as might be advocated by a minority who challenge the status quo).

Figure 1 provides an illustration of such a meritocratic pyramid. It is an organizational form whose participants are usually geographically distributed, and whose work is "scheduled", performed, and evaluated in an autonomous and decentralized manner

(Fielding, Whitehead, *et al.*, 1998, Noll and Scacchi 1999). However, it is neither simply a "cathedral" nor a "bazaar", at least as these attributions have been used and popularized (DiBona, Ockman, and Stone 1999). Instead, when pyramid meritocracy operates as a virtual enterprise, it must rely on *virtual project management* (VPM) to mobilize, coordinate, build, and evaluate (assure quality of) open software development activities. VPM in turn requires multiple people to act in the component sub-roles of a project manager in a manner that may be more ephemeral than persistent.

Virtual project management exists within open software communities to enable control via community decision-making (e.g., voting), Web site and CVS repository administration all possible and effective. Similarly, it exists to mobilize and sustain the use of privately owned resources (e.g., Web servers, network access, site administrator labor, skill and effort) available for reuse by the community.

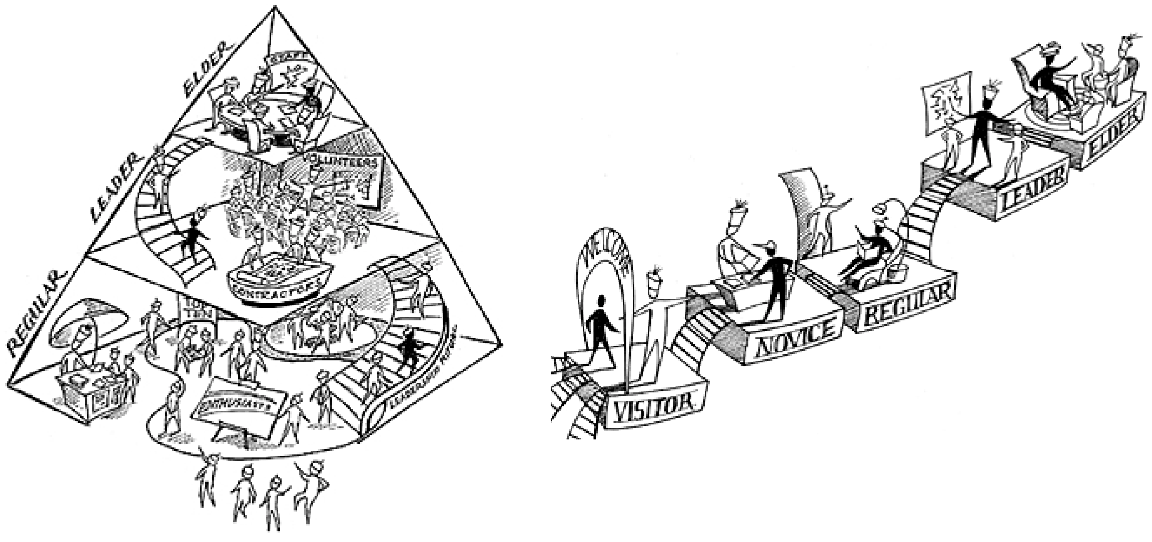

Figure 1. A pyramid meritocracy and role hierarchy (Kim 2000)

Software technology transfer

Software technology transfer is an important and often neglected process within the academic software engineering community. However, in open software communities, the diffusion (distribution), adoption, installation, and routine usage across the Web are all central to the ongoing maintenance of open software systems. Open software technology transfer is a *community building process* that must be institutionalized both within a community and its software to flourish (Kim 2000). In this regard, software technology transfer is not an engineering process (at least, not yet). It is instead socio-technical process that entails the development of constructive social relationships, informally negotiated social agreements, and a commitment to participate through sustained contribution of software discourse and shared representations, much like the other processes identified above. Thus, community building and sustaining participation are essential and recurring activities that enable open software to persist without corporate investment.

Conclusions

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

Acknowledgements

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

References

S. Bendifallah and W. Scacchi, Understanding Software Maintenance Work, *IEEE Trans. Software Engineering*, 13(3), 311-323, March1987.

Boehm, B., A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, Using the WinWin Spiral Model: A Case Study, *Computer*, 31(7), 33-44, 1998.

C. Cleveland, The Past, Present, and Future of PC Mod Development, *Game Developer*, 46-49, February 2000.

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

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

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

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

R.T. Fielding, E.J. Whitehead, K.M. Anderson, G.A. Bolcer, P. Oreizy, and R.N Taylor, Web-based Design of Complex Information Products, *Communications ACM*, 41(8), 84-92, August 1998.

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

J.A. Goguen, Formality and Informality in Requirements Engineering (Keynote Address), *Proc. 4th. Intern. Conf. Requirements Engineering*, 102-108, IEEE Computer Society, 1996.

R.E. Grinter, Supporting Articulation Work Using Configuration Management Systems, *Computer Supported Cooperative Work*, 5(4), 447-465, 1996.

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

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

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

P. Mi and W. Scacchi, Modeling Articulation Work in Software Engineering Processes. *Proc. 1st International Conference on the Software Process*, Redondo Beach, CA, 188-201, IEEE Computer Society, Oct 1991.

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

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

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

J. E. Robbins and D. F. Redmiles, Cognitive support, UML adherence, and XMI interchange in Argo/UML, *Information and Software Technology*, 42(2), 71-149, 25 January 2000.

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

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

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

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

D. Truex, R. Baskerville, and H. Klein, Growing Systems in an Emergent Organization, *Communications ACM*, 42(8), 117-123, 1999.

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

M. Young, *The Rise of the Meritocracy*, 1870-2033, Thames and Hudson, London, 1958.