Free and Open Source Development Practices in the Game Community

Walt Scacchi, *University of California, Irvine*

The free and open source software (FOSS) approach lets communities of like-minded participants develop software systems and related artifacts that are shared freely instead of offered as closed-source commercial products. Free (as in freedom) software and open source software are closely related but slightly different approaches and licensing schemes for developing publicly shared software. Although the amount of popular literature that attests to FOSS is growing,[1,2]

only a small—but growing—number of systematic empirical studies exist that explain how these communities produce software.[3–5] Similarly, little is known about how community participants coordinate software development across different settings, or about what software processes, work practices, and organizational contexts they need for success.

Academic communities, commercial enterprises, and government agencies that want to benefit from FOSS development will need grounded models of its processes and practices

to effectively invest their limited resources. My team at the UC Institute for Software Research investigated the software development practices, social processes, technical system configurations, organizational contexts, and interrelationships that give rise to FOSS systems in different communities. In particular, we looked at the FOSS computer game community to provide examples of common development processes and practices.

Understanding FOSS development practices

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

Researchers are investigating at least four diverse FOSS communities through empirical

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

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

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

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

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

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

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

FOSS development processes

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

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

Requirements analysis and specification

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

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

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

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

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

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

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

Tools such as CVS serve as both a centralized mechanism for coordinating FOSS development and a venue for mediating control over which software enhancements, extensions, or upgrades will be checked in to the archive. If checked in, these updates will be available to the community as part of the alpha, beta, candidate, or official released versions, as well as the daily-build release.
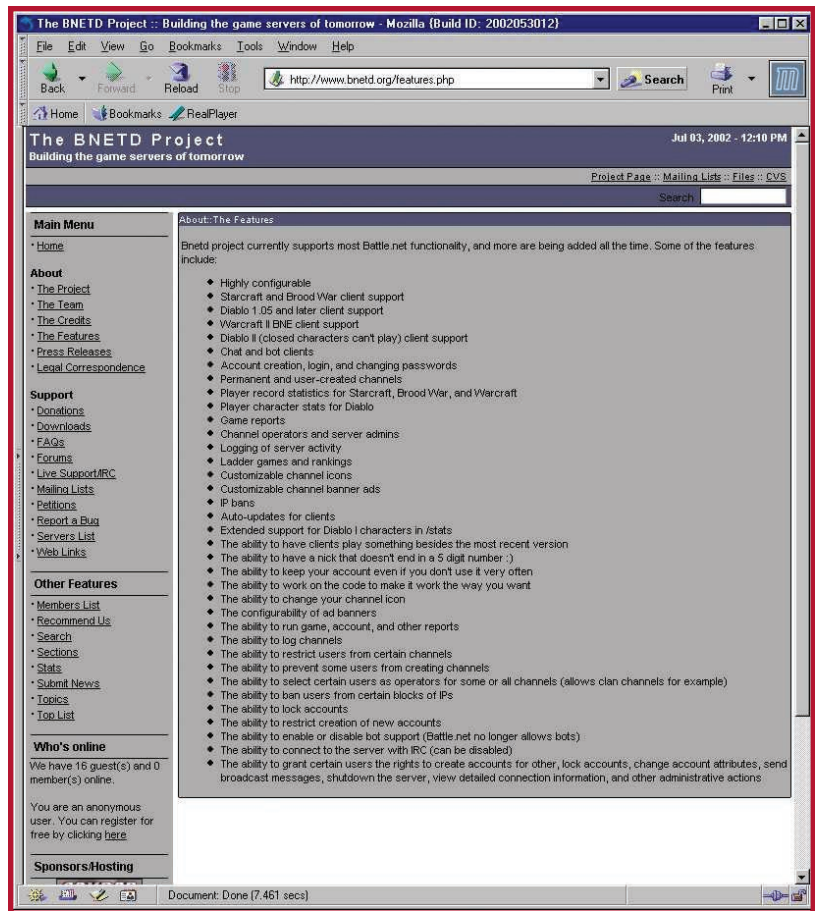
Figure 1. Computer-game software requirements specified as retrospectively asserted features. (figure courtesy of www.bnetd.org, July 2002)

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

Each project team or CVS repository administrator must decide what can be checked in and who can and can't check in new or modified software source code content. Some projects make these policies explicit through a voting scheme,[9] and in other projects they remain informal, implicit, and subject to negotiation with the designated module or version owner. In either case, the team must coordinate version updates for a new system build and release to take place. Subsequently, developers who want to submit updates to the community's shared repository rely primarily on online discussions in *lean media* form, such as threaded email messages posted on a site,[5] rather than having to deal with onerous system configuration control committees or seemingly arbitrary product delivery schedules. So, joint use of ver-
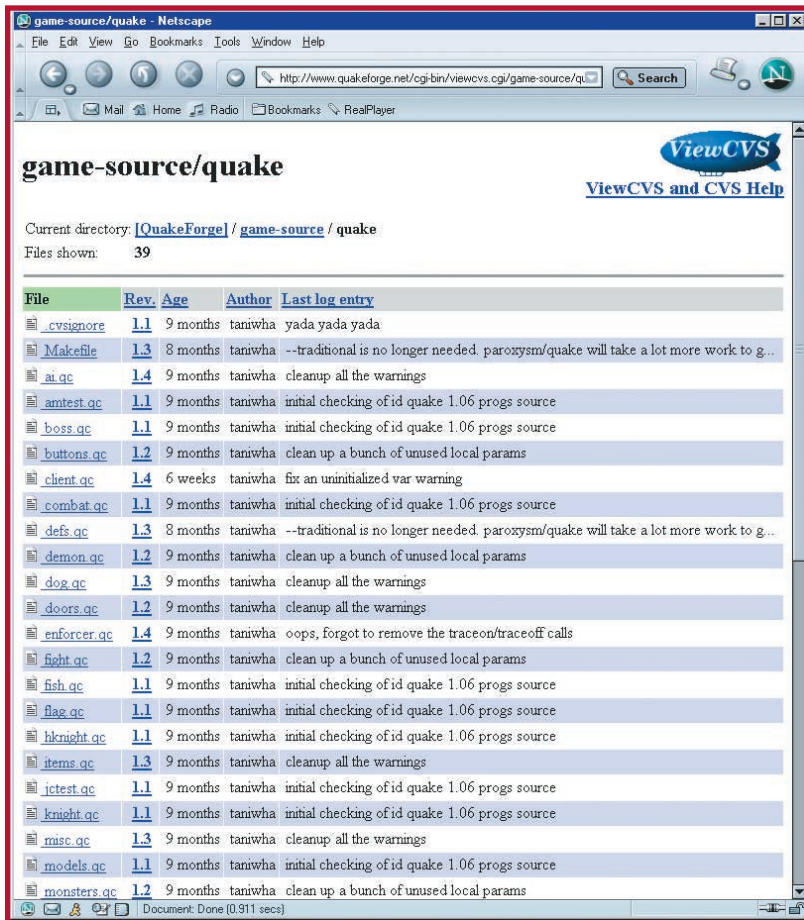
Figure 2. A view into a Web-accessible CVS (Concurrent Versions System) configuration archive of software source code files for the game Quake. (figure courtesy of the Quake-Forge Project)

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

Maintenance as evolutionary redevelopment, reinvention, and revitalization

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

reinvention is a continually emerging source of adaptation, learning, and improvement in FOSS functionality and quality.

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

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

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

Project management and career development

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

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

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

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

VPM exists in FOSS communities to enable effective control via community decision-making and Web site and CVS repository administration. Similarly, it exists to mobilize and sus-
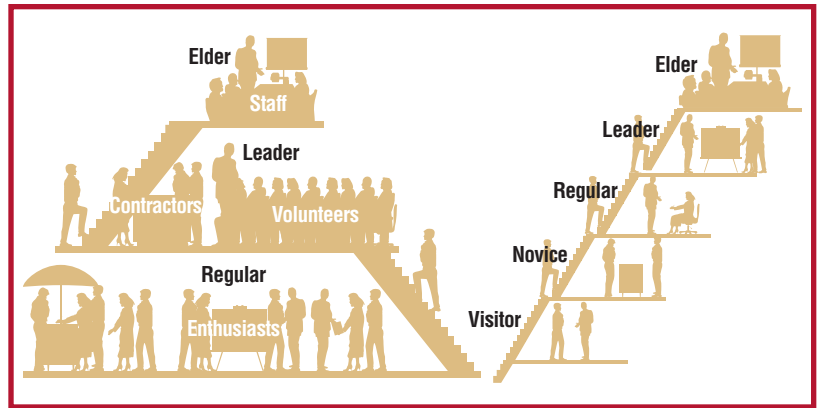
Figure 3. A layered meritocracy and role hierarchy.

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

tain the use of privately owned resources that the community can use (for example, Web servers, network access, site administrator labor, skill, and effort). Finally, some preliminary evidence suggests that, compared to projects with traditional software project management,

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

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

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

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

## Software technology transfer and licensing

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

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

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

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

An overall, essential part of what enables the transfer and practice of FOSS development, and what distinguishes it from traditional software engineering, is the use and reiteration of FOSS public licenses. More than half of the 60,000 FOSS projects registered at Source-Forge use the GNU General Public License. The GPL preserves and reiterates the beliefs and practices of sharing, examining, modifying, and redistributing FOSS systems and assets as property rights for collective freedom. Open source software projects that comingle assets that weren't created as free property have instead adopted variants that relax or strengthen the rights and conditions the GPL lays out. Visit www.opensource.org or www.creativecommons.org for general information on how to create these licenses.
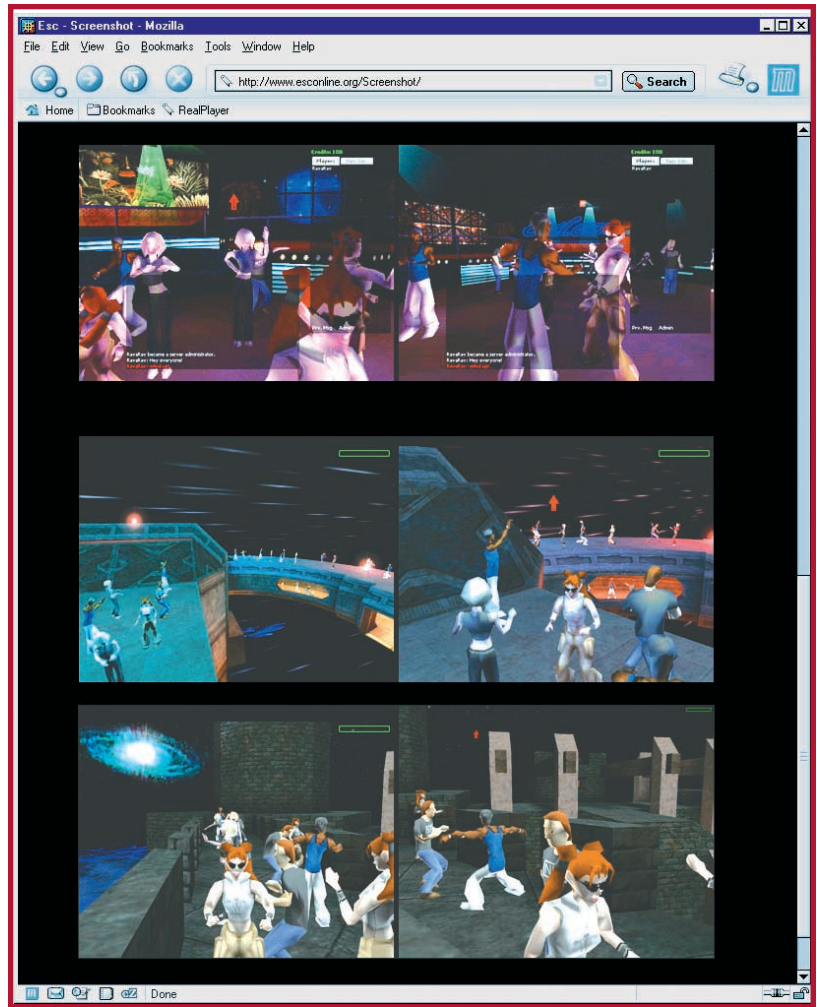
Figure 6. A first-person shooter game (Unreal Tournament) that's been modified and transformed into a 3D virtual environment for socializing and virtual dancing with in-game avatars. (figure courtesy of Martin C. Martin)

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

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

About the Author

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

References

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