Free/Open Source Software Development Practices in the Computer Game Community

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 2003
Revised version to appear in *IEEE Software*

Introduction

This study presents findings from empirical studies of software development practices, social processes, technical system configurations, organizational contexts, and interrelationships that give rise to free or open source[1] software (F/OSS) systems in different communities. F/OSS represents an approach for communities of like-minded participants to develop software systems and related artifacts that are intended to be shared freely, rather than offered as closed commercial products. While there is a growing popular literature attesting to F/OSS [2,11], there are a small but growing number systematic empirical studies that informs how these communities produce software (see **Sidebar** at the end of this document for a study sample). 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 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/OSS development to allow effective investment of their limited resources. Therefore this article investigates processes and practices that arise in F/OSS projects in different communities, and specifically focuses on F/OSS computer game community to provide examples of these common practices.

Understanding F/OSS development practices

There is growing and widespread interest is in understanding the practices and processes of F/OSS development. However, there is no prior model or globally accepted framework that defines how F/OSS is developed in practice. The starting point is thus to investigate F/OSS practices in different communities.

At least four different and diverse F/OSS communities are currently being investigated through empirical studies. These four are centered about the development of software for

[1] Free (as in freedom) software and open source software are closely related but slightly different approaches and licensing schemes for developing software that can be publicly shared.

Internet/Web infrastructure, computer games, software engineering design systems, and X-ray/deep space astronomy.

Rather than examine F/OSS development practices for a single system (e.g., GNU/Linux) which may be interesting but unrepresentative, or related systems from the same community (Internet infrastructure), the focus here is to identify general F/OSS practices both within and across these diverse communities. Thus, the F/OSS development practices that are described below have been empirically observed in different projects in each of these communities. Further, data exhibits in the form of screenshots displaying Web site contents from projects within the computer game community[2] are used to exemplify the practices, though comparable data from the other communities could serve equally well.

Participants within these communities often participate in different *roles* like core developer, module owner, code contributor, code repository administrator, reviewer or end-user. They contribute *software content* (programs, artifacts, execution scripts, code reviews, comments, etc.) to *Web sites* within each community, and communicate information about their content updates via *online discussion forums, threaded email messages*, and *newsgroup postings*. *Screenshots*, *how-to guides*, and lists of *frequently asked questions* are also content that serve to help convey *system use scenarios,* while *software bug reports* appear either in newsgroup messages, bug reporting Web pages, or in bug databases describe what doesn't work during use. Administrators of these sites 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 taxonomic information architecture for *types of site and project-specific content*.

Central to the development of F/OSS in each community are *software extension mechanisms* and F/OSS *software copyright licenses* that insure freedom and/or openness. The extension mechanisms enable modification of the functionality or architecture of software systems via intra-/inter-application scripting or external module plug-in architectures. Copyright licenses, most often derived from the GNU Public License (GPL), are attached to any project developed software, so that it might be further accessed, examined, debated, modified, and redistributed without loss of these rights in the future. These public software licenses stand in contrast to the restricted access found in closed source software systems and licenses. Last, in each of the four communities examined, participants choose on occasion to author and publish *online manuals*, *technical reports or scholarly research papers* about their software development efforts, which are available for subsequent off-line examination and review.

Each of these highlighted items in the preceding paragraphs point to publicly available data that can be collected, analyzed, and represented within narrative ethnographies or computational models of F/OSS development processes. Significant examples of each kind of data have been collected, analyzed, and modeled, and these are described next.

[2] On the SourceForge Web portal, computer games are the *fourth* most popular category of F/OSS projects with more than 7K registered projects, out of 60K.

F/OSS development processes

In contrast to the world of software engineering, F/OSS development communities do not seem to readily adopt or practice modern software engineering processes. F/OSS communities develop software that is extremely valuable, generally reliable, globally distributed, made available for acquisition at little/no cost, and readily used within its associated community. So, what development processes are being routinely used and practiced?

From studies to date, there are at least five types of F/OSS development processes being employed across all four communities. Each process can be briefly described in turn, though none should be construed as being independent or more important than the others. Furthermore, it appears that these processes may occur concurrent to one another, rather than as strictly ordered within a traditional life cycle model, or partially ordered in a spiral model.

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 F/OSS development, how does requirements analysis occur, and where and how are requirements specifications described? Studies to date have yet to report discovery of records for formal requirements elicitation, capture, analysis, and validation activity of the kind suggested by modern software engineering textbooks in any of the four communities under study [9]. In general, they cannot be found online on F/OSS Web sites, or offline in published technical reports or documents identified as "requirements specification" documents with few exceptions. What has been found and observed is different.

It appears at this time that F/OSS 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 contributing new or modified system functionality that effectively creates a system's requirements. F/OSS requirements appear in the form of post hoc assertions within private and public email discussion threads, ad hoc software artifacts (e.g., source code fragments included within a message) and Web site content updates that continually emerge [10]. More conventionally, requirements analysis, specification, and validation do not have first-class status as an assigned or recognized task in F/OSS projects. But what can be observed are widespread practices that imply reading and sense-making of online content, interlinked webs of discourse that effectively traces, condenses and hardens into retrospective software requirements. This arises all while being globally accessible to existing, new or former F/OSS project participants. An example of a *retrospective requirements specification* appears in Figure 1. In short, requirements like these exist because some F/OSS developers implemented their system in such manner, rather than because some user representatives, focus group, or product marketing strategists explicitly specified their need for the identified functionalities.

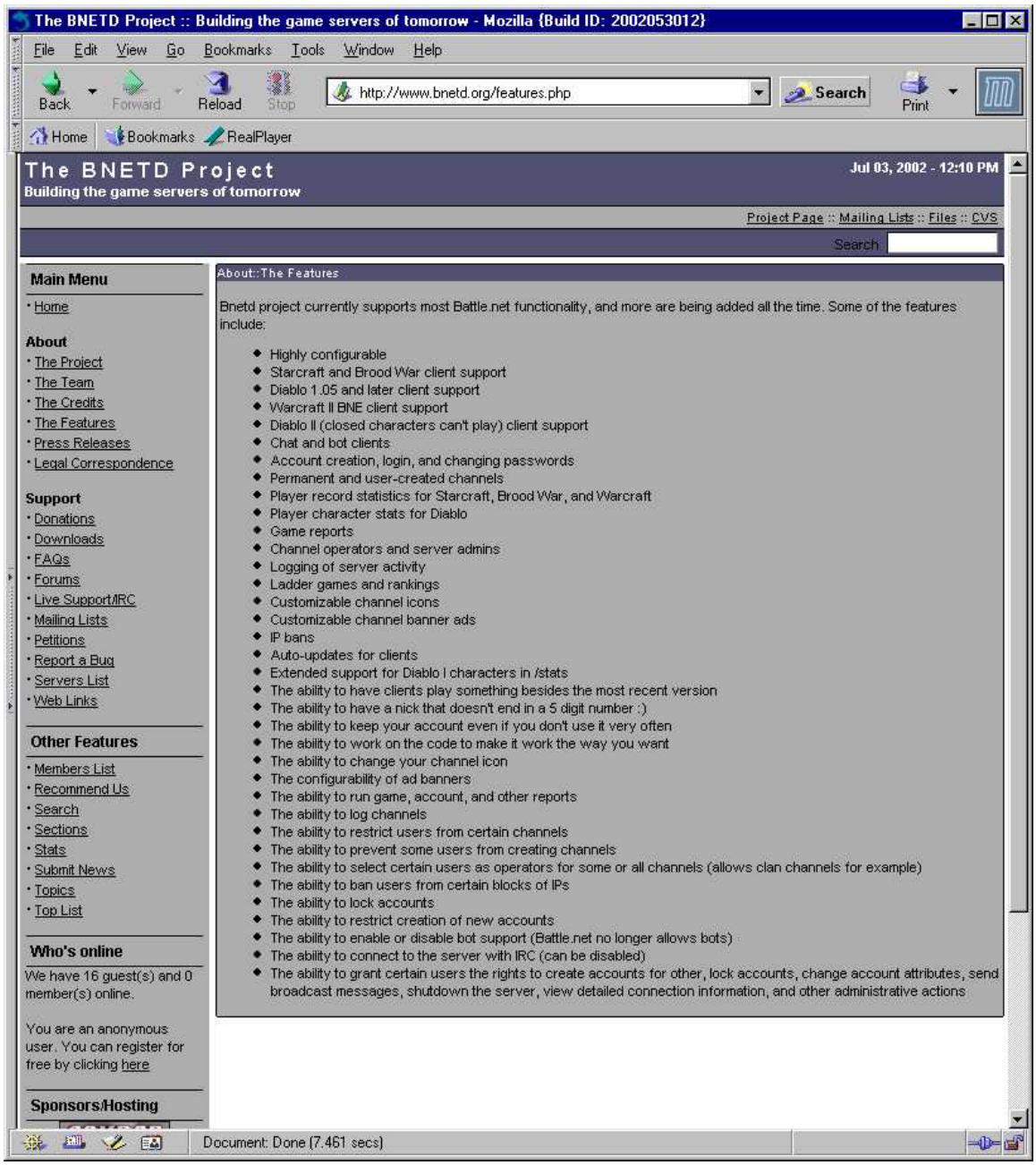

Figure 1. An example of computer game software requirements specified as retrospectively asserted "features" (source: **http://www.bnet.org/features.php**, July 2002)

Coordinated version control, system build, and staged incremental release

Software version control tools such as the concurrent versions system, CVS--itself an F/OSS system and document base [4]--have been widely adopted for use within F/OSS communities. Figure 2 displays a view into one such F/OSS repository on the Web.

Figure 2. A view into a Web accessible CVS configuration archive of software source code files for the game Quake (source: **http://www.quakeforge.net**, December 2002).

Tools like CVS are being used as both a centralized mechanism for coordinating F/OSS development, 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 a software configuration management activity, is a recurring situation that requires coordination but enables stabilization and synchronization of dispersed and somewhat invisible development work. This coordination is required due to the potential

tension between centralized decision-making authority of a project's core developers and decentralized work activity of code contributors when two or more autonomously contributed software system updates are made which overlap, conflict with one another, or generate unwanted side-effects. Each project team, or CVS repository administrator in it, must decide what can be checked in, and who will or will not be able to check-in new or modified software source code content. Sometimes these policies are made explicit through a voting scheme [3], while in others they are left informal, implicit and subject to negotiation. 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 threaded email messages posted on a Web site [12], rather than through onerous or opaque system configuration control boards. Thus, coordinated version control, system build and release is a process that is mediated by the joint use of versioning, system building, and communication tools.

Maintenance as evolutionary redevelopment, reinvention, and redistribution

Software maintenance, in the form of the addition/subtraction of system functionality, debugging, restructuring, tuning, conversion (e.g., internationalization), and migration across platforms, is a widespread, recurring process in F/OSS 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. However, this traditional characterization of software maintenance does not do justice for what can be observed to occur within different F/OSS communities. Instead, it may be better to characterize the overall evolutionary dynamic of F/OSS as *reinvention*. Reinvention is enabled through the sharing, examination, modification, and redistribution of concepts and techniques that have appeared in closed source systems, research and textbook publications, conferences, and the interaction and discourse between developers and users across multiple F/OSS projects. Thus, reinvention is a continually emerging source of improvement in F/OSS functionality and quality.

F/OSS systems seem to evolve through minor improvement or mutations that are expressed, recombined, and redistributed across many releases with short duration life cycles. End-users of F/OSS systems who act as developers or maintainers continually produce these mutations. These mutations appear initially in daily system builds. These modifications or updates are then expressed as a tentative alpha, beta, release candidate, or stable release versions that may survive redistribution and review, then subsequently be recombined and re-expressed with other new mutations in producing a new stable release version. As a result, these mutations articulate and adapt an F/OSS system to what its developer-users want it to do in the course of evolving and continually reinventing the system.

F/OSS systems *co-evolve* with their development communities. This means the evolution of one depends on the evolution of the other. Said differently, a F/OSS project with a small number of developers (most typically one) will not produce and sustain a viable system unless/until the team reaches a larger critical mass of 10-15 core developers.

However, if critical mass is achieved, then it may be possible for the F/OSS system to grow in size and complexity at a sustained exponential rate, defying the laws of software evolution that have held for decades [6].

Last, closed source software systems that were thought to be dead or beyond their useful product life or maintenance period may be *revitalized* through the redistribution and opening of their source code. However, this may only succeed in application domains where there is a devoted community of enthusiastic user-developers who are willing to invest their time and skill to keep the cultural heritage of their former experience with such systems alive. Figure 3 provides an example for vintage arcade games now numbering in the thousands that have been revitalized as F/OSS systems.

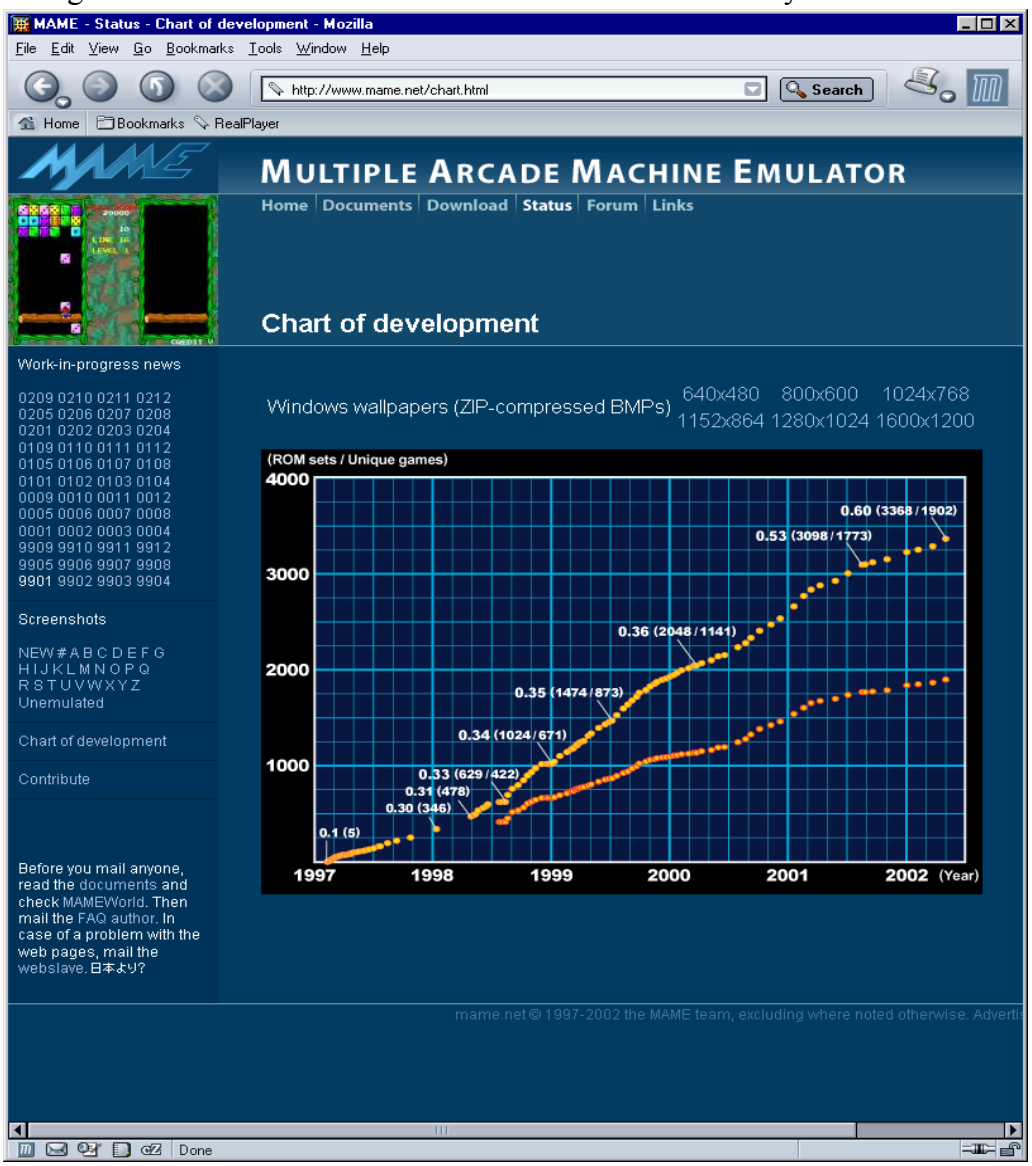

Figure 3. A graphic display depicting sustained growth in the number of vintage arcade ROM sets and games migrated into open source for use on contemporary computer platforms. (source: http://www.mame.net/chart/html, December 2002).

Project management and career development

F/OSS development teams can take the organizational form of a *pyramid meritocracy* [3,5] operating as a dynamically organized virtual enterprise [8]. A pyramid meritocracy 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 does not imply a single authority, since decision-making may be shared among core developers who act as peers at the top echelon of the pyramid. Instead, meritocracies tend to embrace incremental innovations such as evolutionary mutations to an existing software code base over radical innovations. Radical change involves the exploration or adoption of untried or sufficiently different system functionality, architecture, or development methods. Radical software system changes might be advocated by a minority of code contributors who challenge the status quo of the core developers. However, their success in such advocacy usually implies creating and maintaining a separate version of the system, and the potential loss of a critical mass of other F/OSS developers. Thus, incremental mutations tend to win out over time.

Figure 4 illustrates the form of a meritocracy common to many F/OSS projects. In this form, software development work appears to be logically centralized, while being physically distributed in an autonomous and decentralized manner [8]. However, it is neither simply a "cathedral" or a "bazaar" [2]. Instead, when pyramid meritocracy operates as a virtual enterprise, it relies on *virtual project management* (VPM) to mobilize, coordinate, control, build, and assure the quality of F/OSS development activities. It may invite or encourage system contributors to come forward and take a shared, individual responsibility that will serve to benefit the F/OSS collective of user-developers. VPM requires multiple people to act in the roles 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. This need for virtual project management can be seen in the example within Figure 5.

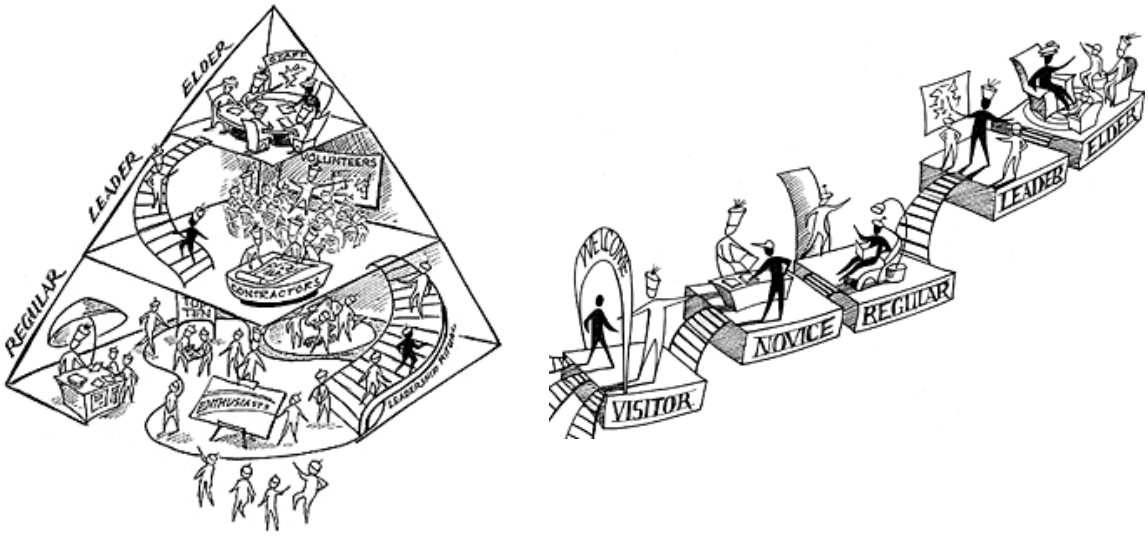

Figure 4. A pyramid meritocracy and role hierarchy [5]

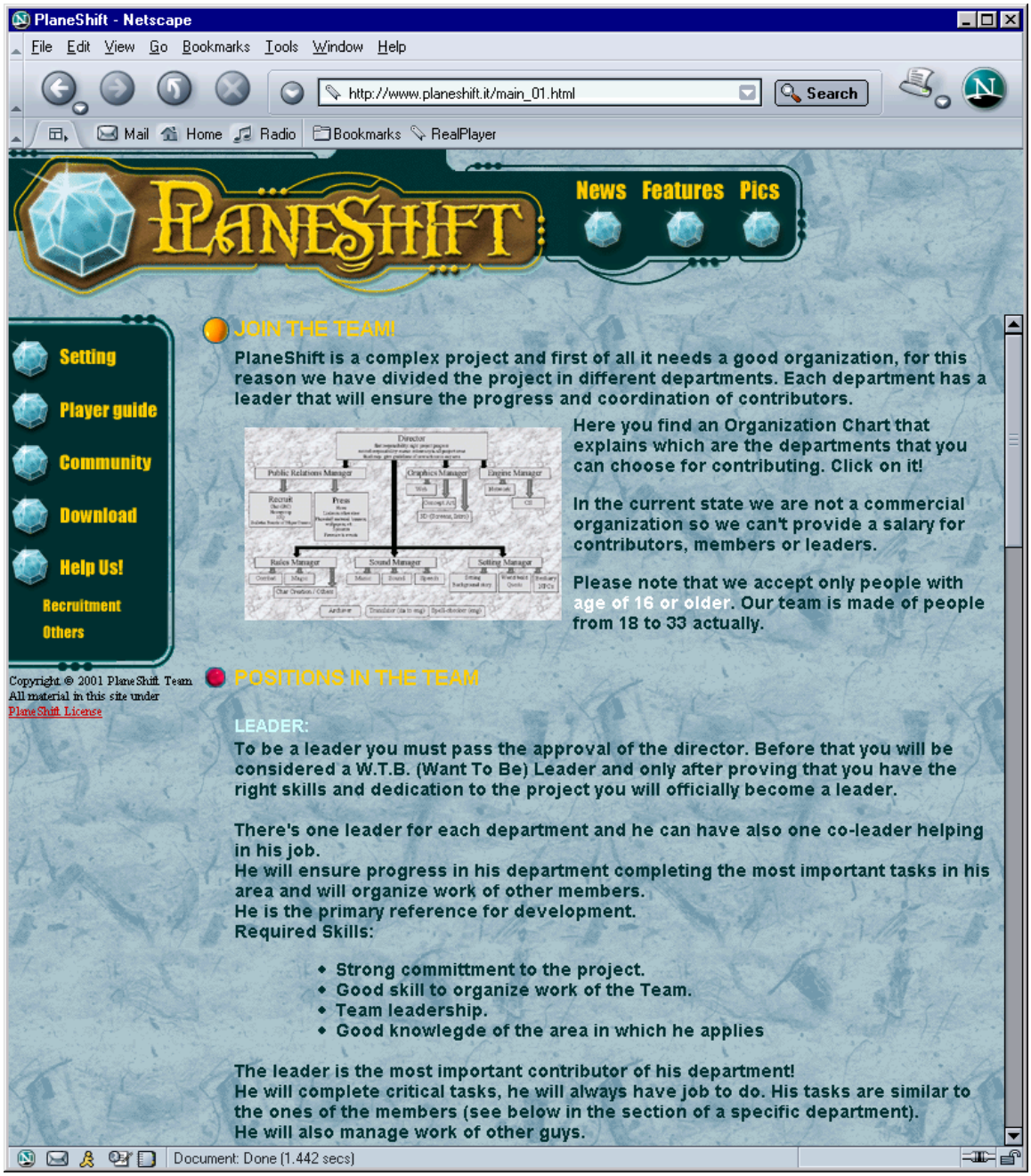

Figure 5. An example statement for how a F/OSS computer game development project seeks to organize and manage itself, using this statement as an organizational surrogate to denote administrative authority, together with an invitation to those who seek such project authority. (Source: **http://www.planeshift.it/main_01.html**, November 2002).

Project participants higher up in the meritocracy have greater perceived authority than those lower down. But these relationships are only effective as long as everyone agrees to their makeup and legitimacy. Administrative or coordination conflicts that cannot be

resolved may end up either by splitting or forking a new system version with the attendant need to henceforth take responsibility for maintaining that version, by reducing one's stake in the ongoing project, or by simply conceding the position in conflict.

Virtual project management exists within F/OSS communities to enable control via community decision-making, Web site and CVS repository administration in an effective manner. Similarly, VPM 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 use or reuse by the community.

Traditional software project management stresses planning and control activities. Lessig [7] on the other hand, observes that source code intentionally or unintentionally realizes a mode of social control on those people who develop or use it. In the case of F/OSS development, Lessig's observation would suggest that the source code controls or constrains end-user and developer interaction, while the code in software development tools, Web sites, and project assets accessible for download controls, constrains, or facilitates developer interaction with the evolving F/OSS system code. CVS is a tool that enables some form of social control. However, the fact that the source code to these systems are available in a free and open source manner, offers the opportunity to examine, revise, and redistribute patterns of social control and interaction in ways that favor one form of project organization, system configuration control, and user-developer interaction over others. Beyond this, the ability for the eyes of many developers to review or look over source code, system build and preliminary test results, and response to bug reports also realizes peer review and the potential for embarrassment as a form of indirect social control over the timely actions of contributing F/OSS developers. Thus, F/OSS development allows for this dimension of VPM to be open for manipulation by the core developers, so as to encourage certain patterns of software development and social control, and to discourage others that may not advance the collective needs of F/OSS project participants.

Last, there are complex motivations for why F/OSS developers are willing to allocate their time, skill, and effort to the ongoing evolution of their systems. Sometimes they may simply see their effort as something that is fun, personally rewarding, or provides a venue where they can exercise and improve their technical competence in a manner that may not be possible with their current job or line of work. In the case of F/OSS for computer games, "…people even get hired for doing these things…", as indicated in the example shown in Figure 6. Furthermore, becoming a central node in a social network of software developers that interconnects multiple F/OSS projects is also a way to accumulate social capital and recognition from peers. However, it also enables the merger of independent F/OSS systems into larger composite ones that gain the critical mass of core developers to grow more substantially and attract ever larger user-developer communities.

Figure 6. An example in the center highlighting career development opportunities for would-be computer game developers via open source game mods (source: http://www.unrealtournament.com/editing, Dec 2002).

Software technology transfer and licensing

Software technology transfer is an important and often neglected process within the academic software engineering community. However, in F/OSS communities, the diffusion, adoption, installation, and routine usage of F/OSS software systems and their Web-based assets, are all central to the ongoing evolution of F/OSS systems. The transfer of F/OSS technology from existing Web sites to organizational practice is a *community and project team building process* [5]. Adoption and use of F/OSS project Web sites are a community wide practice for how to publicize and share F/OSS project assets. These Web sites can be built using F/OSS Web site content management systems (e.g., PhP-Nuke) to host project contents that can be served using F/OSS Web servers (Apache), database systems (MySQL) or application servers (JBoss), and increasingly accessed via F/OSS Web browsers (Mozilla). Furthermore, ongoing F/OSS projects may employ dozens of F/OSS development tools, whether as standalone systems like CVS, as integrated development environments like NetBeans or Eclipse, or as sub-system components of their own F/OSS application in development. These projects similarly employ asynchronous systems for project communications that are persistent, searchable, traceable, public and globally accessible.

F/OSS 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, a routine willingness to search, browse, download and try out F/OSS assets from a variety of projects and Web sites. It is also a commitment to participate through sustained contribution of public, Web-based discourse and shared representations about F/OSS systems, much like the other processes identified above. Thus, community building and sustaining participation are essential and recurring activities that enable F/OSS to persist without centrally planned and managed corporate software development centers.

F/OSS systems, development assets, tools, and project Web sites serve as a venue for socializing, building relationships and trust, sharing and learning with others. Some OSS projects have taken developing such social relationships as their primary project goal. The system depicted in Figure 7 is an example, where developers took an existing networked game system and created an open source modification [1] to it that transformed it into a venue for social activity.

Last, an overall and essential part of what enables the transfer and practice of F/OSS development, and what distinguishes it from traditional software engineering, is the use and reiteration of F/OSS public licenses. More than half of the 60K F/OSS projects registered at SourceForce employ the GPL for free (as in freedom) software. The GPL seeks to preserve and reiterate the beliefs and practices of sharing, examining, modifying and redistributing F/OSS systems and assets as property rights for collective freedom. Other OSS projects, because of the co-mingling of assets that were not created as free property, have instead adopted variants that relax or strengthen the rights and conditions laid out in the GPL. An example of such a variant appears in Figure 8.

Figure 7. Example of a first-person shooter game (Unreal Tournament) that has been modified and transformed into a 3D virtual environment for socializing and virtual dancing with in-game avatars (source: **http://www.esconline.org/**, December 2002).

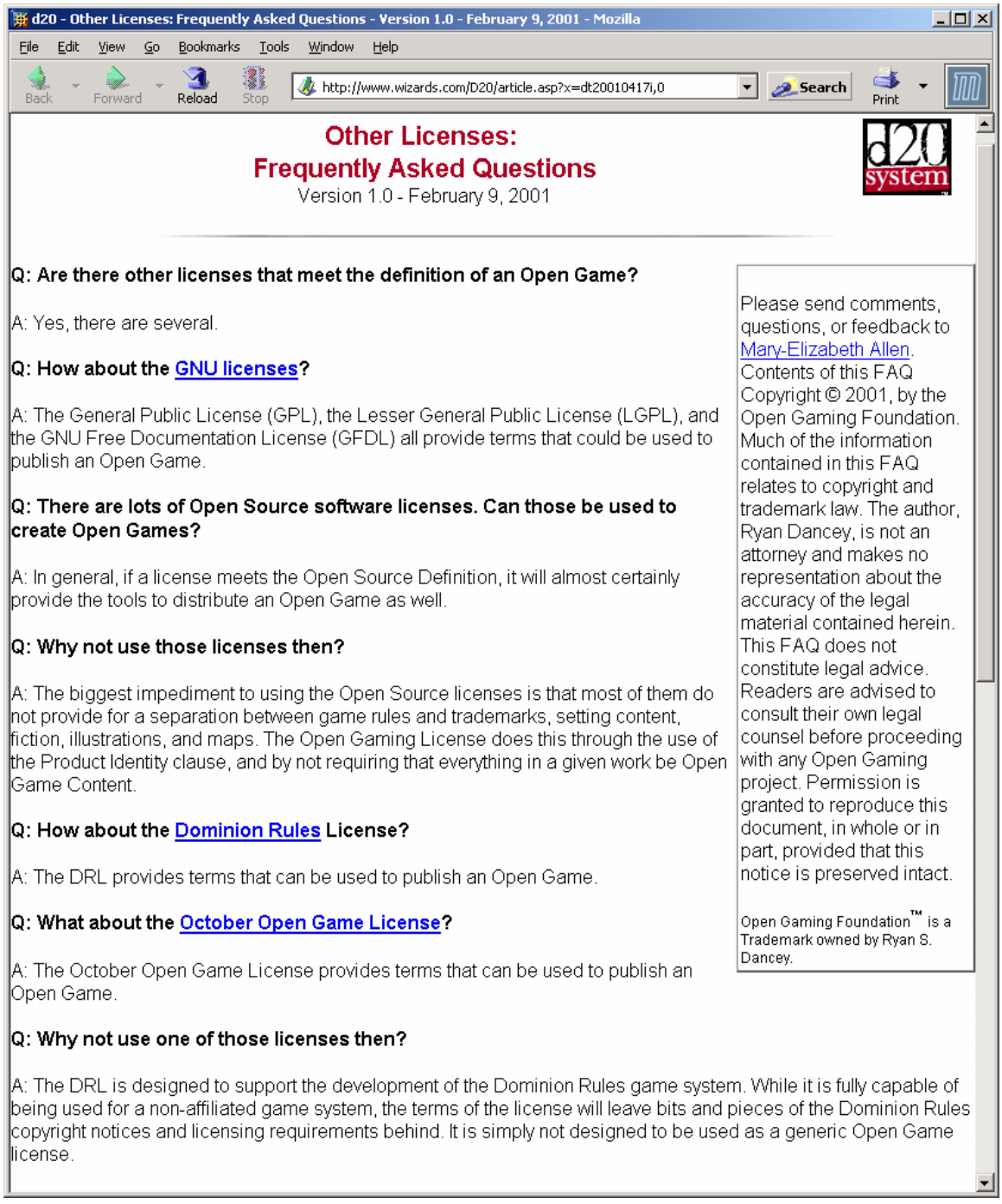

Figure 8. An example of an open license insuring software redistribution and modification freedoms like the GPL, as well as other rights specific to computer games (source: http:www.wizards.com/D20/, February 2003)

Conclusions

Free or open source software development practices are giving rise to a new view of how complex software systems can be constructed, deployed, and evolved. F/OSS development does not adhere to the traditional rationality found in the legacy of software engineering life cycle models or prescriptive standards. F/OSS development is inherently a complex web of socio-technical processes, development situations, and dynamically emerging development contexts. This paper provides results from empirical studies that begin to outline some of the processes and practices for how F/OSS systems are developed in different communities. In particular, examples drawn from the world of computer games reveal how processes and practices for the development and propagation of F/OSS technology are intertwined and mutually situated to the benefit of those motivated to use and contribute to it.

Acknowledgements

The research described in this report is supported by grants #IIS-0083075, #ITR-0205679 and #ITR-0205724 from the National Science Foundation. No endorsement implied.

References

1. C. Cleveland, The Past, Present, and Future of PC Mod Development, *Game Developer*, 46-49, February 2001.

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

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

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

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

6. M.M. Lehman, Programs, Life Cycles, and Laws of Software Evolution, *Proc. IEEE*, 68, 1060-1078, 1980.

7. L. Lessig, *CODE and Other Laws of Cyberspace*, Basic Books, New York, 1999.

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

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

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

11. S. Williams, *Free as in Freedom: Richard Stallman's Crusade for Free Software*, O'Reilly and Associates, Sebastopol, CA, 2002.

12. 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.

Biography: Walt Scacchi is 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, at the University of California Irvine. He received his Ph.D. in Information and Computer Science at UC Irvine in 1981. He joined ISR in 1999 after serving on the faculty at the University of Southern California for 18 years. From 1981 to 1991 he founded and directed the USC System Factory, and from 1993 to 1998 he directed the USC ATRIUM Laboratory. His interests include open source software development, software process engineering, software acquisition and electronic commerce, and organizational studies of system development. He is an active researcher with more than 100 research papers, and consults widely to clients in industry and government agencies.

Sidebar: *Empirical studies of F/OSS development*

Compared to the study of software engineering practices in different settings, empirical study of F/OSS projects are in relatively small number, and focus almost exclusively on software systems for Internet/Web applications or the GNU/Linux operating system. Nonetheless, these studies pose interesting research challenges and reveal some surprising results.

As described elsewhere in this article, many of the largest and most popular F/OSS systems like the Linux Kernel [3,14], GNU/Linux distributions [4,11], GNOME user interface [7] and others are growing at an exponential rate, as is their internal architectural complexity [14]. On the other hand the vast majority of F/OSS projects are small, short-lived, exhibit little/no growth, and often only involve the effort of one developer [1,6,8]. In this way, the overall trend derived from samples of 400-40K F/OSS projects registered at the SourceForge (www.sourceforge.net) Web portal reveals a power law distribution common to large self-organizing systems. This means a few large projects have a critical mass of at least 10-15 core F/OSS developers [9] and inevitably garner the most attention, software downloads, and usage. The vast majority, on the other hand, are small F/OSS projects unlikely to thrive and grow. But what is significant about this overall population of projects and developers is that as many as 60% or more developers participate in two or more projects, and on the order of 5% participate in 10 or more F/OSS projects [5]. These have been labeled "linchpin developers" [8] to indicate their role in enabling previously independent small F/OSS projects to 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 this trend is found in traditional or closed source software projects is unclear.

Also possibly significant are findings appearing in multiple independent studies that reveal it is common in F/OSS projects to find end-users as developers and developers as end-users [9,10,12,15]. Similarly, user-developer communities co-evolve with their systems in a mutually dependent manner [2,10,11,12], and system architectures and functionality growing in discontinuous jumps as independent F/OSS projects decide to join forces [3,10,12,13]. Again, whether this trend is found in traditional or closed source software projects is unclear. But what these findings and trends do indicate is that it appears that the *practice* of F/OSS development is different from that advocated in traditional software engineering.

Sidebar References

1. Capilupppi, A., P. Lago, and M. Morisio, Evidences in the Evolution of OS projects through Changelog Analyses, *Proc. 3rd Workshop on Open Source Software Engineering,* Portland, OR, May 2003.

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

3. M.W. Godfrey and Q. Tu, Evolution in Open Source Software: A Case Study, *Proc. 2000 Intern. Conf. Software Maintenance* (ICSM-00), San Jose, CA, October 2000.

4. J.M Gonzalez-Barahona, M.A. Ortuno Perez, P. de las Heras Quiros, J. Centeno Gonzalez, and V. Matellan Olivera, Counting Potatoes: The Size of Debian 2.2, *Upgrade Magazine*, II(6), 60-66, December 2001.

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

6. F. Hunt and P. Johnson, On the Pareto Distribution of SourceForge Projects, in C. Gacek and B. Arief (eds.), *Proc. Open Source Software Development Workshop*, 122-129, Newcastle, UK, February 2002.

7. S. Koch and G. Schneider, Effort, Co-operation and Co-ordination in an Open Source Software Project: GNOME, *Info. Sys. J.,* 12(1), 27-42, 2002.

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

9. A. Mockus, R.T. Fielding, and J. Herbsleb, Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Trans. Software Engineering and Methodology*, 11(3), 309-346, 2002.

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

11. S. O'Mahony, 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, 2003.

12. W. Scacchi, Understanding the Requirements for Development Open Source Software Systems, *IEEE Proceedings – Software*, 149(1), 24-39, February 2002.

13. W. Scacchi, *Open EC/B: A Case Study in Electronic Commerce and Open Source Software Development*, technical report, Institute for Software Research, July 2002.

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

15. E. von Hippel and R. Katz, Shifting Innovation to Users via Toolkits, *Management Science*, 48(7), 821-833, July 2002.