

Socio-Technical Interaction Networks in Free/Open Source Software Development Processes

Walt Scacchi
Institute for Software Research
School of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA
+1-949-824-4130 (v), +1-949-824-1715 (f)
Wscacchi@uci.edu

September 2004
Previous version: May 2004

Revised version to appear in S.T. Acuña and N. Juristo (eds.), *Software Process Modeling*, pp. 1-27, Springer Science+Business Media Inc., New York, 2005.

This chapter explores patterns of social and technological interaction that emerge in free/open source software development (F/OSSD) projects found in different research and development communities. F/OSSD is a relatively new way for building and deploying large software systems on a global basis, and differs in many interesting ways from the principles and practices traditionally advocated for software engineering. Hundreds of F/OSS systems are now in use by thousands to millions of end-users, and some of these F/OSS systems entail hundreds-of-thousands to millions of lines of source code. So what's going on here, and how are F/OSSD processes that are being used to build and sustain these projects different?

One of the more significant features of F/OSSD is the formation and enactment of complex software development processes performed by loosely coordinated software developers and contributors. These people may volunteer their time and skill to such effort, and may only work at their personal discretion rather than as assigned and scheduled. Further, these developers generally provide their own computing resources, and bring their own software development tools with them. Similarly, F/OSS developers work on software projects that do not typically have a corporate owner or management staff to organize, direct, monitor, and improve the software development processes being put into practice on such projects. But how are successful F/OSSD projects and software development processes possible without regularly employed and scheduled software development staff, or without an explicit regime for software engineering project management? Why will software developers participate in F/OSSD projects? Why and how are large F/OSSD projects sustained? How are large F/OSSD projects coordinated, controlled or managed without a traditional project management team? Why and how might these answers to these questions change over time? These are the core research questions that will be addressed in this chapter.

Socio-technical interaction networks (STINs) are an emerging conceptual framework for identifying, organizing, and comparatively analyzing patterns of social interaction, system development, and the configuration of components that constitute an information system. More specifically, a STIN denotes a set of collective relationships among:

“...people (including organizations), equipment, data, diverse resources (money, skill, status), documents and messages, legal arrangements and enforcement mechanisms, and resource flows. The elements of a STIN are heterogeneous. The network relationships between these elements include social, economic, and political interactions.” [Kling 2003]

Subsequently, STINs provide a scheme for examining the networks of people who work together through interrelated social and technical processes that arise to create the complex information systems and products. STINs thus serve as a conceptual framework through which to examine ongoing F/OSSD projects and processes.

STINs may be seen as the conceptual outgrowth of what historically was called “socio-technical systems” (STS) [Emery 1960], informed by “actor network theory”. An STS perspective envisions a world of complex organizations that routinely employ technicians/engineers to develop systems for users, where success in developing a system depends on the participation and sustained involvement of the system’s users. If people issues in the design, deployment, and evolution of these STS are slighted or ignored, then these systems would be problematic or unsatisfactory to use, else be outright failures. However, understanding this pathology, or intervening to prevent it, is possible through STS practices that can be incorporated into system development processes [Scacchi 2004c]. Historically, STS design approaches prescriptively advocated user involvement and participation in the design and deployment of information systems, and its successors like “participatory design” [Schuler 1993] advocate more up-to-date renditions of STS design. Consequently, STS design was among the earliest approaches to system development that sought to both engage and balance the interests of people (developers, end-users), products (systems, documentation, etc.) and processes (system design and usage) in a manner that focused on participation and involvement of all system stakeholders. Other directions for advancing STS design include its integration with workplace democracy movements [Bjerknes 1995, Ehn 1987] and soft systems approaches [Atkinson 2000], as well as its reconstitution as a customer-driven system design method [Beyer 1997].

Actor-network theory (ANT) [cf. Callon 1986, Latour 1987, Law 1999] on the other hand draws attention to processes by which scientific disputes or technical design alternative become closed and rationalized, ideas accepted, tools and methods adopted, or more simply how decisions are made about what is known. ANT does not assume or encourage prescriptive strategies or motives for why people should participate or be involved in system design. Instead, it draws attention to need for empirical study of what people do in their work, and what tools, resources, and artifacts they produce, use, or consume along the way. Furthermore, ANT draws attention to the relationships that repeatedly emerge in the ways people in different roles and with different resources in overlapping settings

articulate scientific research or system development processes through situated work practices.

STINs build on concepts from STS design and ANT by drawing attention to the web of relationships that interlink what people do in the course of their system development work to the resources they engage and to the products (software components, development artifacts, and documents) they create, manipulate, and sustain. STINs thus give us a way to better observe the contexts in which people carry out software development processes and related work practices. In F/OSSD projects, this web is manifest and articulated over the World-Wide Web and associated systems for creating and updating the web, so that it can be observed, navigated, and empirically studied. Introducing and explaining how STINs appear in different F/OSSD projects, is therefore part of the purpose of this chapter. In turn, STINs are then used as a framework to observe and focus on why and how software developers participate in F/OSSD projects, what sustains their interest and communities, how participation and community gives rise to socio-technical conditions that serve to coordinate and control F/OSSD processes and practices, and how and why they evolve over time.

This chapter seeks to explore and develop answers to questions about F/OSSD by examining the patterns and networks of interactions among the people, products, and processes that are found in a growing base of empirical studies of F/OSSD projects. Exhibits from a variety of different F/OSSD projects will be presented and used to empirically ground the analysis and findings to be presented in this chapter.

Understanding F/OSS development practices and processes

There is growing and widespread interest 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 [Mockus 2002, Scacchi 2002, 2004]. The starting point is thus to investigate F/OSS practices in different communities.

F/OSSD projects are being empirically studied in at least six different and diverse F/OSS communities. These six are centered about the development of software for Internet/Web infrastructure, computer games, electronic business/commerce applications, academic support software, software engineering design systems, and X-ray/deep space astronomy.

Rather than examine F/OSSD practices for a single system (e.g., Linux kernel) which may be interesting but unrepresentative of most F/OSSD projects, or of related systems from the just one community (e.g., Internet infrastructure), the focus here is to identify general F/OSS practices shaped by STINs 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 across the different F/OSS project communities are used to exemplify the practices, though comparable data from a different selection of F/OSS projects could serve equally well.

From studies to date, there are at least four areas where the formation and activity of STINs is most apparent across F/OSSD projects within and across all six communities. These include (a) participating, joining, and contributing to F/OSS projects; (b) forming alliances and building communities of practice through linked artifacts; (c) coordinating, cooperating, and controlling F/OSSD projects; and (d) co-evolving social and technical systems for F/OSS. Each can be briefly described in turn, though none should be construed as being independent or more important than the others. Furthermore, it appears that each can occur concurrent to one another, rather than as strictly ordered within a traditional life cycle model, or partially ordered in a spiral model.

Participating, Joining, and Contributing in F/OSS projects

There are complex motivations for why F/OSS developers are willing to allocate their time, skill, and effort by joining a F/OSS project [Hars 2002, Hertel 2003, von Krogh 2003]. 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 within their current job or line of work. However, people who participate, contribute, and join F/OSS projects tend to act in ways where building trust and reputation [Stewart 2001], achieving “geek fame” [Pavlicek 2000], being creative [Fischer 2001], as well as giving and being generous with one’s time, expertise, and source code [Bergquist 2001] are valued traits. In the case of F/OSS for software engineering design systems, participating in such a project is a viable way to maintain or improve software development skills, as indicated in Exhibit 1.

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. One survey reports that 60% or more F/OSS developers participate in two or more projects, and on the order of 5% participate in 10 or more F/OSS projects [Hars 2002]. In addition, participation in F/OSS projects as a core developer can realize financial rewards in terms of higher salaries for conventional software development jobs [Hann 2002, Lerner 2002]. 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 [Madey 2004, Scacchi 2004c].

People who participate in F/OSS projects do so within one or more roles. Gacek and Arief [Gacek 2004] provide a common classification of the hierarchy of roles that people take and common tasks they perform when participating in a F/OSS project, as shown in Figure 1. Typically, it appears that people join a project and specialize in a role (or multiple roles) they find personally comfortable and intrinsically motivating [von Krogh 2004]. In contrast to traditional software development projects, there is no explicit assignment of developers to roles, though individual F/OSSD projects often post guidelines or “help wanted here” for what roles for potential contributors are in greatest need.

It is common in F/OSS projects to find end-users becoming contributors or developers, and developers acting as end-users [Mockus 2002, Nakakoji 2002, Scacchi 2002, von

Hippel 2002]. As most F/OSS developers are themselves end-users of the software systems they build, they may have an occupational incentive and vested interest in making sure their systems are really useful. However the vast majority of participants probably simply prefer to be users of F/OSS systems, unless or until their usage motivates them to act through some sort of contribution. Avid users with sufficient technical skills may actually work their way through each of the roles and eventually become a core developer, as suggested by Figure 2. As a consequence, participants within F/OSS project often participate in different roles within both technical and social networks [Smith 1999, Preece 2000] in the course of developing, using, and evolving F/OSS systems.

Making contributions is often a prerequisite for advancing technically and socially within a community, as is being recognized by other community members as having made substantive contributions [Fielding 1999, Kim 2000]. Most commonly, F/OSS project participants contribute different types of software representations or content (source code, bug reports, design diagrams, execution scripts, code reviews, test case data, Web pages, email comments, online chat, etc.) to Web sites of the F/OSS projects they join. The contribution—the authoring, hypertext linking (when needed), and posting/uploading—of different types of content helps to constitute an ecology [Erickson 2000, Spinuzzi 2000] of software informalisms [Scacchi 2002] that is specific to a F/OSS project, though individual content types are widely used across most F/OSS projects. Similarly, the particular mix of software informalisms employed by participants on a F/OSS project articulates an information infrastructure [Star 1996] for framing and solving problems that arise in the ongoing development, deployment, use, and support of the F/OSS system at the center of a project.

Administrators of open software community Web sites and source code repositories serve as gatekeepers in the choices they make for what information to post, when and where within the site to post it, as well as what not to post [Smith 1999]. Similarly, they may choose to create a site map that constitutes a classification of site and domain content, as well as community structure and boundaries [O'Mahony 2003].

Most frequently, participants in F/OSS projects engage in online discussion forums or threaded email messages as a central way to observe, participate in, and contribute to public discussions of topics of interest to community participants [Yamauchi 2000]. However, these people also engage in private online or offline discussions that do not get posted or publicly disclosed, due to their perceived sensitive content.

Central to the development of F/OSS projects 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 end-user license agreements.

Finally, in each of the six communities being examined, 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, review, and secondary analysis.

Forming alliances and building community through participation, artifacts, and tools

How does the gathering of individual F/OSS developers give rise to a more persistent project team or self-sustaining community? Through choices that developers make for their participation and contribution to an F/OSSD project, they find that there are like-minded individuals who also choose to participate and contribute to a project. These software developers find and connect with each other through F/OSSD Web sites and online discourse (e.g., threaded email discussions) [Monge 1998], and they find they share many technical competencies, values, and beliefs in common [Crowston 2002, Espinosa 2002, Elliott 2004]. This manifests itself in the emergence of an occupational network of F/OSS developers [Elliott 2003].

Sharing beliefs, values, communications, artifacts and tools among F/OSS developers enables not only cooperation, but also provides a basis for shared experience, camaraderie, and learning [cf. Brown 1991, Fischer 2001, George 1995]. F/OSS developers participate and contribute by choice, rather than by assignment, since they find that conventional software development work provides the experience of working with others who are assigned to a development effort, whether or not they find that share technical approaches, skills, competencies, beliefs or values. As a result, F/OSS developers find they get to work with people that share their many values and beliefs in common, at least as far as software development. Further, the values and beliefs associated with free software or open source software are both signaled and institutionalized in the choice of intellectual property licenses (e.g., GPL) that F/OSSD projects adopt and advocate. These licenses in turn help establish norms for developing free software or open source software, as well as for an alliance with other F/OSSD projects that use the same licenses.

More than half of the 80K F/OSS projects registered at SourceForce.net Web portal employ the GNU General Public License (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. A few large F/OSSD project that seek to further protect the collective free/open intellectual property rights do so through the formation of legally constituted non-profit organizations or foundations (e.g., Free Software Foundation, Apache Software Foundation, GNOME Foundation) [O'Mahony 2003]. Other OSS projects, because of the co-mingling of assets that were not created as free property, have adopted variants that relax or strengthen the rights and conditions laid out in the GPL. Dozens of these licenses now exist, with new ones continuing to appear (cf. www.opensource.org). An example of

such a variant appears in Exhibit 2. Finally, when OSSD projects seek to engage or receive corporate sponsorship, and the possible co-mingling of corporate/proprietary intellectual property, then some variation of a non-GPL open source license is employed, as a way to signal a “business friendly” OSSD project, and thus to encourage participation by developers who want to work in such a business friendly and career enhancing project [Hann 2002, Sharma 2002].

Developing F/OSS systems is a community and project team building process that must be institutionalized within a community [Sharma 2002, Smith 1999, Preece 2000] for its software informalisms (artifacts) and tools to flourish. Downloading, installing, and using F/OSS systems acquired from other F/OSS Web sites is also part of a community building process [Kim 2000]. 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 the software version control system 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 systems, hyperlinked artifacts and tools, and project Web sites serve as venues for socializing, building relationships and trust, sharing and learning with others. “Linchpin developers” [Madey 2004] act as community forming hubs that enable independent small F/OSS projects to come together as a larger social network with the critical mass [Marwell 1993] 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. F/OSSD Web sites also serve as hubs that centralize attention for what is happening with the development of the focal F/OSS system, its status, participants and contributors, discourse on pending/future needs, etc. Furthermore, by their very nature, these Web sites (those accessible outside of a corporate firewall) are generally global in reach and accessibility. This means the potential exists for contributors to come from multiple remote sites (geographic dispersion) at different times (24/7), from multiple nations, representing the interests of multiple cultures or ethnicity.

All of these conditions point to new kinds of requirements—for example, community building requirements, community software requirements, and community information sharing system (Web site and interlinked communication channels for email, forums, and chat) requirements. These requirements may entail both functional and non-functional requirements, but they will most typically be expressed using open software informalisms, rather than using formal notations based on some system of mathematical logic.

Community building, alliance forming, and participatory contributing are essential and recurring activities that enable F/OSSD projects to persist without central corporate authority. Figure 3 depicts an example of a social network of 24 F/OSS developers within 5 F/OSS projects that are interconnected through two linchpin developers [Madey 2004]. Thus, linking people, systems, and projects together through shared artifacts and sustained online discourse enables a sustained socio-technical community, information infrastructure [Star 1996], and network of alliances [Kling 2003, Monge 1998] to emerge.

Cooperating, coordinating, and controlling F/OSS projects

Getting software developers to work together, even when they desire to cooperate is not without its challenges for coordinating and controlling who does what when, and to what they do it to. Conflicts arise in both F/OSSD [Elliott 2003, Elliott 2004, Jensen 2004] and traditional software development projects [Sawyer 2001], and finding ways to resolve conflicts becomes part of the cost (in terms of social capital) that must be incurred by F/OSS developers for development progress to occur. Minimizing the occurrence, duration, and invested effort in such conflicts quickly becomes a goal for the core developers in an F/OSSD project. Similarly, finding tools and project organizational forms that minimize or mitigate recurring types of conflicts also becomes a goal for experienced core developers.

Software version control tools such as the concurrent versions system CVS--itself an F/OSS system and document base [Fogel 1999]--have been widely adopted for use within F/OSS projects. Tools like CVS are being used as both a centralized mechanism for coordinating and synchronizing 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 for check-out 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 [Grinter 1996]. 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 project contributors when two or more autonomously contributed software source code/content updates are made which overlap, conflict with one another, or generate unwanted side-effects [Grinter 2003]. It is also practiced as a way to manage, track, and control both desired and undesired dependencies within the source code [deSouza 2003], as well as among its surrounding informalisms [Scacchi 2002, 2004]. Tools like CVS thus serve to help manage or mitigate conflicts over who gets to modify what, at least as far as what changes or updates get included in the next software release from a project. However, the CVS administrator or configuration control policies provide ultimate authority and control mediated through such systems.

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

[Fielding 1999], 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 online discussions that are supported using "lean media" such as threaded email messages posted on a Web site [Yamauchi 2000], rather than through onerous system configuration control boards. Thus, software version control, system build and release is a coordination and control process mediated by the joint use of versioning, system building, and communication tools [Erenkrantz 2003].

F/OSSD projects teams can take the organizational form of a *layered meritocracy* [Fielding 1999, Kim 2000] operating as a dynamically organized virtual enterprise [Crowston 2002, Noll 1999]. A layered 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 layer.

Figure 2 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 [Noll 1999]. However, it is neither simply a "cathedral" or a "bazaar", as these terms have been used to describe alternative ways of organizing software development projects. Instead, when layered 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 implied requirement for virtual project management can be seen in the text appearing within Exhibit 3.

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 administration, 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 shared use or collective reuse by the community.

Traditional software project management stresses planning and control activities. In contrast, Lessig and others [Lessig 1999, Shah 2003] observe that source code is an institution for collective action [O'Mahony 2003, Ostrom 1990] that 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 is 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 responses 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/OSSD 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/OSSD project participants. Subsequently, F/OSSD projects are managed, coordinated and controlled, though without the roles for traditional software project managers.

Co-evolving socio-technical systems for F/OSS

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, as well as also a collective approach to organizational learning in F/OSS projects [Brown 1991, Fischer 2001, Huntley 2003, George 1995].

Many of the largest and most popular F/OSS systems like the Linux Kernel [Godfrey 2000, Schach 2002], GNU/Linux distributions [Gonzalez-Barahona 2001, O'Mahony 2003], GNOME user interface [Koch 2002] and others are growing at an exponential rate, as is their internal architectural complexity [Schach 2002]. 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 [Capiluppi 2003, Madey 2004]. In this way, the

overall trend derived from samples of 400-40K F/OSS projects registered at the 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 5-15 core F/OSS developers [Mockus 2002] that act in or share project leadership roles [Fielding 1999] that are surrounded by dozens to hundreds of other contributors, and hundreds to millions of end users. These F/OSS projects that attain and sustain such critical mass are those that inevitably garner the most attention, software downloads, and usage. On the other hand, the vast majority of F/OSS projects are small, lacking in critical mass, and thus unlikely to thrive and grow.

The layered meritocracies that arise in F/OSS projects 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.

F/OSS systems seem to evolve through minor improvements 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.

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. Exhibit 4 provides an example for vintage arcade games now numbering in the thousands that are being revitalized and evolved through F/OSS systems.

Overall, F/OSS systems co-evolve with their development communities. This means the evolution of one depends on the evolution of the other. Said differently, an 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 5-15 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 [Lehman 1980, Scacchi 2004b]. Furthermore, user-developer communities co-evolve with their systems in a mutually dependent manner [Elliott 2004, Nakakoji 2002, O'Mahony 2003, Scacchi 2002], and system architectures and functionality grow in discontinuous jumps as independent F/OSS projects decide to join forces [Godfrey 2000, Nakakoji 2002, Scacchi 2002b]. 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 processes is different from the processes traditionally advocated for software engineering.

Limitations and Constraints of STINs on F/OSS Development Processes

F/OSS is certainly not a panacea for developing complex software systems, nor is it simply software engineering done poorly. Instead, it represents an alternative community-intensive approach to develop software systems and related artifacts, as well as social relationships. However, it is not without its limitations and constraints. Thus, we should be able to help see these limits as manifest within or through STINs for each of the four types of processes examined above.

First, in terms of participating, joining, and contributing to F/OSS projects, a developer's interest, motivation, and commitment to a project and its contributors is dynamic and not indefinite. F/OSS developers are loathe to find themselves contributing to a project that is realizing commercial or financial benefits that are not available to all contributors, or that are concentrated to benefit a particular company, again without some share going to the contributors. Some form of reciprocity seems necessary to sustain participation, whereas a perception of exploitation by others can quickly dissolve a participant's commitment to further contribute, or worse to dissuade other participants to abandon an open source project that has gone astray. If linchpin developers lose interest, then unless another contributor comes forward to fill in or take over role and responsibility for the communication and coordination activities of such key developers, then the F/OSS system may quickly become brittle, fragile, and difficult to maintain. Thus, participation, joining, and contributing must become sustained activities on an ongoing basis within F/OSS projects for them to succeed.

Second, in terms of forming alliances and building community through participation, artifacts, and tools points to a growing dependence on other F/OSS projects. The emergence of non-profit foundations that were established to protect the property rights of large multi-component F/OSS project creates a demand to sustain and protect such foundations. If a foundation becomes too bureaucratic as a result to streamline its operations, then this may drive contributors away from a project. So, these foundations need to stay lean, and not become a source of occupational careers, in order to survive and evolve. Similarly, as F/OSS projects give rise to new types of requirements for community building, community software, and community information sharing systems, these requirements need to be addressed and managed by F/OSS project contributors in roles above and beyond those involved in enhancing the source code of a F/OSS project. F/OSS alliances and communities depend on a rich and growing web of socio-technical relations. Thus, if such a web begins to come apart, or if the new requirements cannot be

embraced and satisfied, then the F/OSS project community and its alliances will begin to come apart.

Third, in terms of cooperation, coordination, and control, F/OSS projects do not escape conflicts in technical decision-making, or in choices of who gets to work on what, or who gets to modify and update what. As F/OSS projects generally lack traditional project managers, then they must become self-reliant in their ability to mitigate and resolve outstanding conflicts and disagreements. Beliefs and values that shape system design choices, as well as choices over which software tools to use, and which software artifacts to produce or use, are determined through negotiation rather than administrative assignment. Negotiation and conflict management then become part of the cost that F/OSS developers must bear in order for them to have their beliefs and values fulfilled. It is also part of the cost they bear in convincing and negotiating with others often through electronic communications to adopt their beliefs and values. Time, effort, and attention spent in negotiation and conflict management are not spent building and improving source code, but they do represent an investment in building and sustaining a negotiated socio-technical network of dependencies.

Last, in terms of the co-evolution of F/OSS systems and community, as already noted, individual and shared resources of people's time, effort, attention, skill, sentiment (beliefs and values), and computing resources are part of the socio-technical web of F/OSS. Reinventing existing software systems as F/OSS coincides with the emergence or reinvention of a community who seeks to make such system reinvention occur. F/OSS systems are common pool resources [Ostrom 1990] that require collective action for their development, mobilization, use, and evolution. Without the collective action of the F/OSS project community, the common pool will dry up, and without the common pool, the community begins to fragment and disappear, perhaps to search for another pool elsewhere.

Conclusions

Free/open source software development practices are giving rise to a new view of how complex software systems can be constructed, deployed, and evolved on a global basis. 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 interaction networks. This paper examines and analyzes results from empirical studies that begin to outline some of the socio-technical activities that situate how F/OSS systems are developed in different communities. In particular, examples drawn from different F/OSS project communities 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.

The future of research in the development and use of STINs as a conceptual framework for observing and analyzing F/OSSD processes and practices seems likely to focus attention to the following topics.

First, the focus of software process research is evolving to include attention to socio-technical processes of people, resources, organizational forms, and institutional rules that embed and surround an F/OSS system, as well as how they interact and interface with one another. Such a focus draws attention to the web of socio-technical relations that interlink people in particular settings to a situated configuration of globally available Web-based artifacts and locally available resources (skills, time, effort, computing) that must collectively be mobilized or brought into alignment in order for a useful F/OSS system to be continuously (re)designed to meet evolving user needs.

Second, participation in F/OSS system design, assertion of system requirements, or design decision-making is determined by effort, willingness, and prior public experience in similar situations, rather than by assignment by management or some other administrative authority. Similarly, the openness of the source code/content of a F/OSS system encourages and enables many forms of transparency, access, and ability to customize/localize a system's design to best address user/developer needs in a particular site or installation.

Third, people who participate in the development, deployment, and evolution of F/OSS often do it on a voluntary or self-selected basis. These people quickly recognize the need to find ways to cooperate and collaborate in order to minimize individual effort and conflict while maximizing collective accomplishment. This is most easily observed in the online (or Web-based) communications, shared source code files and directories, application invocation or system configuration scripts, Web pages and embedded hyperlinks, and other textual artifacts that people in free/open source software project communities employ as the media, content, and (hyperlinked) context of system design and evolution. However, there is a continually emerging need to minimize and mitigate conflicts that arise in F/OSSD projects due to the absence of a traditional project management regime that might otherwise act to competently resolve (or to incompetently bungle) such software development conflicts. As a result, F/OSSD projects have adapted or evolved the use of tools, interlinked artifacts, and organizational forms that effectively create a project management capability and socio-technical control framework without (traditional) project managers.

Fourth, the world of F/OSSD is different in many interesting ways and means when compared to the world of software engineering within corporate or centralized enterprise settings. Knowing and understanding one does not provide a sufficient basis for assuming an understanding of the other, yet both worlds develop complex software systems and artifacts using development processes that may (or may not) be well understood. This analysis of the socio-technical interaction networks that facilitate and constrain F/OSSD processes and practices points to new concepts, situations, events, and data for understanding how large software systems are developed, deployed, and evolved within F/OSSD communities of practice. Each merits further study, articulation, and refinement.

Last, the four preceding research directions collectively begin to draw attention to matters beyond software development processes, as traditionally addressed. Instead, future STIN and software process research can employ Web analyses [Kling 1982, Kling 2003],

ethnographic methods [Elliott 2004, Scacchi 2002, Viller 2000] and contemporary socio-technical system design techniques [Scacchi 2004c] to study and model how people accomplish software development processes and practices in an organizational setting using F/OSS systems, artifacts, tools, people, and circumstances at hand. Understanding the F/OSS system or interaction network will need to include understanding the workplace, inter-organizational networks, social worlds and cultural milieu that embed and situate how people interact with and through the F/OSS systems at hand in the course of their work and workflows. Similarly, there is a basic need to discover new ways and means that enable traditional software developers to understand and become users of F/OSSD practices so as to empower and sustain both traditional and F/OSS developers in their collective effort to continuously improve their software development skills, practices, and processes. This chapter therefore represents a step in this direction.

Acknowledgements: The research described in this report is supported by grants #ITR-0083075, #ITR-0205679, #ITR-0205724, and #ITR-0350754 from the U.S. National Science Foundation. No endorsement implied. Mark Ackerman at University of Michigan, Ann Arbor; Les Gasser at University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; and Margaret Elliott and Chris Jensen at the UCI Institute for Software Research are collaborators on the research described in this chapter.

References

Atkinson, C.J., Socio-Technical and Soft Approaches to Information Requirements Elicitation in the Post-Methodology Era, *Requirements Engineering*, 5, 67-73, 2000.

Bjerknes, G. and Bratteteig, T., User Participation and Democracy. A Discussion of Scandinavian Research on System Development. *Scandinavian Journal of Information Systems*. 7(1), 73-98, 1995.

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

Beyer, H. and Holtzblatt, K., *Contextual Design: A Customer-Centered Approach to Systems Designs*, Morgan Kaufmann Publishers, San Francisco, CA, 1997.

Brown, J.S. and Duguid, P., Organizational learning and communities-of-practice: Toward a unified view of working, learning, and innovation. *Organization Science*, 2(1):40-57, 1991.

Callon, M., Law, J., and Rip, J., (eds.), *Mapping the Dynamics of Science and Technology: Sociology of Science in the Real World*. London: Macmillan Press, 1986.

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

Crowston, K., Annabi, H., and Howison, J., Defining Open Source Software Project Success, *Proc. 24th Intern. Conf. Information Systems (ICIS-2003)*, December 2003.

Crowston, K., and Scozzi, B., Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development, *IEE Proceedings--Software*, 149(1), 3-17, 2002.

Ehn, P. and Kyng, M., The Collective Resource Approach to System Design, in G. Bjerknes, P. Ehn, and M. Kyng (eds.), *Computers and Democracy—a Scandinavian Challenge*, Avebury, Aldershot, 1987.

Emery, F.E. and Trist, E.L., Socio-Technical Systems. In C.W. Churchman & M. Verhurst (Eds), *Management Science, Models and Techniques*, Vol. 2, 83-97. London: Pergamon Press, 1960.

Elliott, M. and Scacchi, W., Free Software Developers as an Occupational Community: Resolving Conflicts and Fostering Collaboration, *Proc. ACM Intern. Conf. Supporting Group Work*, 21-30, Sanibel Island, FL, November 2003.

Elliott, M. and Scacchi, W., Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture, in S. Koch (ed.), *Free/Open Source Software Development*, Idea Publishing, to appear, 2004.

Erenkrantz, J., Release Management within Open Source Projects, *Proc. 3rd. Workshop on Open Source Software Engineering*, 25th. Intern. Conf. Software Engineering, Portland, OR, May 2003.

Erickson, T., Making Sense of Computer-Mediated Communication (CMC): CMC Systems as Genre Ecologies, *Proc. 33rd Hawaii Intern. Conf. Systems Sciences*, IEEE Press, 1-10, January 2000.

Espinosa, J. A., Kraut, R.E., Slaughter, S. A., Lerch, J. F., Herbsleb, J. D., Mockus, A. Shared Mental Models, Familiarity, and Coordination: A multi-method study of distributed software teams. *Intern. Conf. Information Systems*, 425-433, Barcelona, Spain, December 2002.

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

Fischer, G., External and shareable artifacts as opportunities for social creativity in communities of interest, in J. S. Gero and M. L. Maher (eds), *Proc. Computational and Cognitive Models of Creative Design*, 67-89, Heron Island, Australia, December 2001.

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

Gacek, C. and Arief, B., The Many Meanings of Open Source, *IEEE Software*, 21(1), 34-40, January/February 2004.

George, J.F., Iacono, S., and Kling, R., Learning in Context: Extensively Computerized Work Groups as Communities-of-Practice, *Accounting, Management and Information Technology*, 5(3/4):185-202, 1995.

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

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

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

Grinter, R.E., Recomposition: Coordinating a Web of Software Dependencies, *Computer Supported Cooperative Work*, 12(3), 297-327, 2003.

Hann, I-H., Roberts, J., Slaughter, S., and Fielding, R., Economic Incentives for Participating in Open Source Software Projects, in *Proc. Twenty-Third Intern. Conf. Information Systems*, 365-372, December 2002.

Hars, A. and Ou, S., Working for Free? Motivations for participating in open source projects, *Intern. J. Electronic Commerce*, 6(3), 2002.

Hertel, G., Neidner, S., and Hermann, S., Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel, *Research Policy*, 32(7), 1159-1177, July 2003.

Huntley, C.L., Organizational Learning in Open-Source Software Projects: An Analysis of Debugging Data, *IEEE Trans. Engineering Management*, 50(4), 485-493, 2003.

Jensen, C. and Scacchi, W., Collaboration, Leadership, and Conflict Negotiation in the NetBeans.org Community, *Proc. 4th Workshop on Open Source Software Engineering*, Edinburgh, UK, May 2004.

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

Kling, R., Kim, G., and King, R., A Bit More to IT: Scholarly Communication Forums as Socio-Technical Interaction Networks, *Journal American Society for Information Science and Technology*, 54(1), 47-67, 2003.

- Kling, R. and Scacchi, W. The Web of Computing: Computer Technology as Social Organization, in A. Yovits (ed.), *Advances in Computers*, 21, Academic Press, 3-85, 1982.
- Koch, S. and Schneider, G., Effort, Co-operation and Co-ordination in an Open Source Software Project: GNOME, *Info. Sys. J.*, 12(1), 27-42, 2002.
- Latour, B., *Science in Action*, Cambridge, MA, Harvard University Press, 1987.
- Law, J. and Hassard, J., (eds.), *Actor Network Theory and After*, Blackwell Publishers, 1999.
- Lehman, M.M., Programs, Life Cycles, and Laws of Software Evolution, *Proc. IEEE*, 68, 1060-1078, 1980.
- Lerner, J. and Tirole, J., Some Simple Economics of Open Source, *J. Industrial Economics*, 50(2), 197-234, 2002.
- Lessig, L., *CODE and Other Laws of Cyberspace*, Basic Books, New York, 1999.
- Madey, G., Freeh, V., and Tynan, R., Modeling the F/OSS Community: A Quantative Investigation, in Koch, S., (ed.), *Free/Open Source Software Development*, Idea Publishing, to appear, 2004.
- Marwell, G. and Oliver, P., *The Critical Mass in Collective Action: A Micro-Social Theory*. Cambridge University Press, 1993.
- Mockus, A., Fielding, R., & Herbsleb, J.D., Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology*, 11(3), 309-346, 2002.
- Monge, P.R., Fulk, J., Kalman, M.E., Flanagan, A.J., Parnassa, C., and Rumsey, S., Production of Collective Action in Alliance-Based Interorganizational Communication and Information Systems, *Organization Science*, 9(3), 411-433, 1998.
- Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., and Ye, Y., Evolution Patterns of Open-Source Software Systems and Communities, *Proc. 2002 Intern. Workshop Principles of Software Evolution*, 76-85, 2002.
- Noll, J. and Scacchi, W., Supporting Software Development in Virtual Enterprises, *J. Digital Information*, 1(4), February 1999.
- O'Mahony, S., Guarding the Commons: How community managed software projects protect their work, *Research Policy*, 32(7), 1179-1198, July 2003.

O'Mahony, S., 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, 2004.

Ostrom, E., Calvert, R., and T. Eggertsson (eds.), *Governing the Commons: The Evolution of Institutions for Collective Action*, Cambridge University Press, 1990.

Paulson, J.W., Succi, G., and Eberlein, A., An Empirical Study of Open-Source and Closed-Source Software Products, *IEEE Trans. Software Engineering*, 30(4), 246-256, April 2004.

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

Preece, J., *Online Communities: Designing Usability, Supporting Sociability*. Chichester, UK: John Wiley & Sons, 2000.

Sawyer, S., Effects of intra-group conflict on packaged software development team performance, *Information Systems J.*, 11, 155-178, 2001.

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

Scacchi, W., Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 21(1), 59-67, January/February 2004a.

Scacchi, W., Understanding Free/Open Source Software Evolution, in N.H. Madhavji, M.M. Lehman, J.F. Ramil and D. Perry (eds.), *Software Evolution*, John Wiley and Sons Inc, New York, to appear, 2004b.

Scacchi, W., Socio-Technical Design, to appear in W. S. Bainbridge (ed.), *The Encyclopedia of Human-Computer Interaction*, Berkshire Publishing Group, 2004c.

Schuler, D. and Namioka, A.E., *Participatory Design: Principles and Practices*, Mahwah, NJ, Lawrence Erlbaum Associates, 1993.

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

Shah, R.C. and Kesan, J.P., Manipulating the governance characteristics of code, *Info*, 5(4), 3-9, 2003.

Sharma, S., Sugumaran, and Rajagopalan, B., A Framework for Creating Hybrid Open-Source Software Communities, *Information Systems J.*, 12(1), 7-25, 2002.

Sim, S.E. and Holt, R.C., “The Ramp-Up Problem in Software Projects: A Case Study of How Software Immigrants Naturalize,” *Proc. 20th Intern. Conf. Software Engineering*, Kyoto, Japan, 361-370, 19-25 April, 1998.

Smith, M. and Kollock, P. (eds.), *Communities in Cyberspace*, Routledge, London, 1999.

Spinuzzi, C. and Zachry, M., Genre Ecologies: An open-system approach to understanding and constructing documentation, *J. Computer Documentation*, 24(3), 169-181, 2000.

de Souza, C.R.B., Redmiles, D., Mark, G., Penix, J. and Sierhuis, M., Management of interdependencies in collaborative software development, *Proc. 2003 Intern. Symp. Empirical Software Engineering (ISESE 2003)*, IEEE Computer Society, 294–303, 2003.

Star, S.L. and Ruhleder, K., Steps Toward an Ecology of Infrastructure: Design and Access for Large Information Spaces, *Information Systems Research*, 7(1), 111-134, March 1996.

Stewart, K.J. and Gosain, S., An Exploratory Study of Ideology and Trust in Open Source Development Groups, *Proc. 22nd Intern. Conf. Information Systems (ICIS-2001)*, in New Orleans, LA. 2001.

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

Viller, S. and Sommerville, I., Ethnographically informed analysis for software engineers, *Intern. J. Human-Computer Studies*, 53, 169-196, 2000.

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

von Krogh, G., Spaeth, S., and Lakhani, K., Community, joining, and specialization in open source software innovation: a case study, *Research Policy*, 32(7), 1217-1241, July 2003.

Yamauchi, Y., Yokozawa, M., Shinohara, T., and Ishida, T., 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.

Tigris.org
Open Source Software Engineering

User: Password: [Login](#)
[Register](#) | [Login help](#)

[My pages](#) [Projects](#) [Community](#)

Search
 [Go](#)
Advanced search
POWERED BY **COLLABNET™**

How do I... [?](#)
· [Get help?](#)

Category	Featured projects
scm	Subversion, RapidSVN, TortoiseSVN
issuetrack	Scarab
requirements	xmlbasedsrs
design	ArgoUML
techcomm	eyebrowse, binarycloud
construction	phpcreate, lptools
testing	maxq, aut
deployment	current
process	ReadySET
libraries	GEF, Axion, Style, SSTree
profession	readings, spin
students	elmuth, ankhsvn

Tigris.org Newsletter
Get the latest news and feature articles.
· [Newsletter info](#)
· [Mar 2004 issue](#)

Tigris.org Community Scope

- Tigris.org is a mid-sized open source community focused on building better tools for collaborative software development.
- You will not find thousands of unrelated projects here: every project fits into the Tigris mission.
- You will not find dead projects here: every project is welcomed into the community with a commitment to see it through and active developers cycle among related projects.
- Tigris.org is hosted by CollabNet, but the Tigris mission is one for the entire open source movement and one that has attracted senior open source developers from many organizations.

The Tigris Mission: Promoting Open Source Software Engineering

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.

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 a 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.

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 a great for developers to keep their skills current and plug into a growing base of shared experience for everyone in the field.

Invitation to contributors Start your project in this fertile valley

Exhibit 1. An example near the bottom highlighting career/skill development opportunities arising from participation in F/OSS projects (source: <http://www.tigris.org/> March 2004).

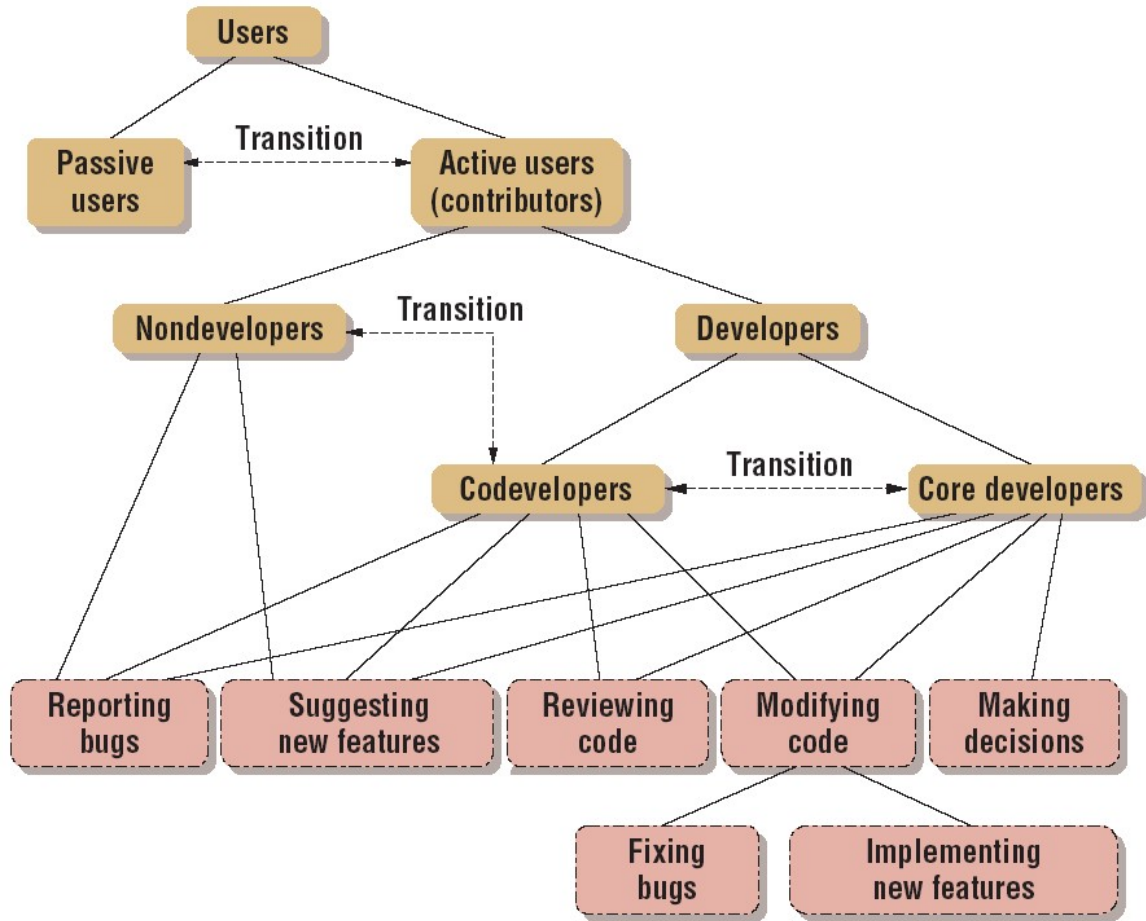
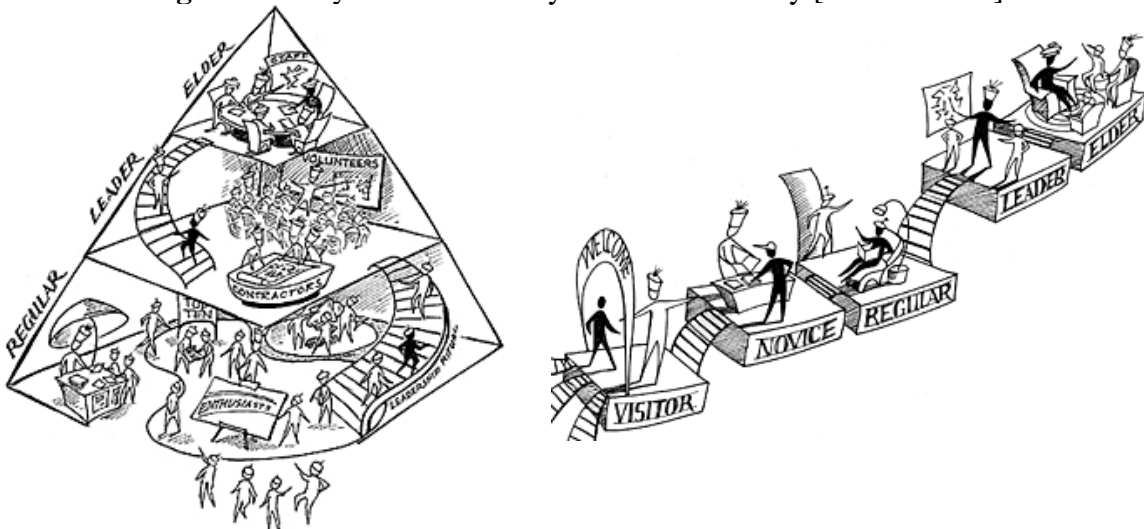


Figure 1. A classification of roles and associated activities that contributing F/OSS participants can perform [Gacek 2004].

Figure 2. A layered meritocracy and role hierarchy [cf. Kim 2000].




d20 - Other Licenses: Frequently Asked Questions - Version 1.0 - February 9, 2001 - Mozilla

File Edit View Go Bookmarks Tools Window Help

Back Forward Reload Stop <http://www.wizards.com/D20/article.asp?x=dt20010417i,0> Search Print

Other Licenses: Frequently Asked Questions

Version 1.0 - February 9, 2001



Q: Are there other licenses that meet the definition of an Open Game?

A: Yes, there are several.

Q: How about the [GNU licenses](#)?

A: The General Public License (GPL), the Lesser General Public License (LGPL), and the GNU Free Documentation License (GFDL) all provide terms that could be used to publish an Open Game.

Q: There are lots of Open Source software licenses. Can those be used to create Open Games?

A: In general, if a license meets the Open Source Definition, it will almost certainly provide the tools to distribute an Open Game as well.

Q: Why not use those licenses then?

A: The biggest impediment to using the Open Source licenses is that most of them do not provide for a separation between game rules and trademarks, setting content, fiction, illustrations, and maps. The Open Gaming License does this through the use of the Product Identity clause, and by not requiring that everything in a given work be Open Game Content.

Q: How about the [Dominion Rules License](#)?

A: The DRL provides terms that can be used to publish an Open Game.

Q: What about the [October Open Game License](#)?

A: The October Open Game License provides terms that can be used to publish an Open Game.

Q: Why not use one of those licenses then?

A: The DRL is designed to support the development of the Dominion Rules game system. While it is fully capable of being used for a non-affiliated game system, the terms of the license will leave bits and pieces of the Dominion Rules copyright notices and licensing requirements behind. It is simply not designed to be used as a generic Open Game license.

Please send comments, questions, or feedback to [Mary-Elizabeth Allen](#).
 Contents of this FAQ Copyright © 2001, by the Open Gaming Foundation. Much of the information contained in this FAQ relates to copyright and trademark law. The author, Ryan Dancey, is not an attorney and makes no representation about the accuracy of the legal material contained herein. This FAQ does not constitute legal advice. Readers are advised to consult their own legal counsel before proceeding with any Open Gaming project. Permission is granted to reproduce this document, in whole or in part, provided that this notice is preserved intact.

Open Gaming Foundation™ is a Trademark owned by Ryan S. Dancey.

Exhibit 2. 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).

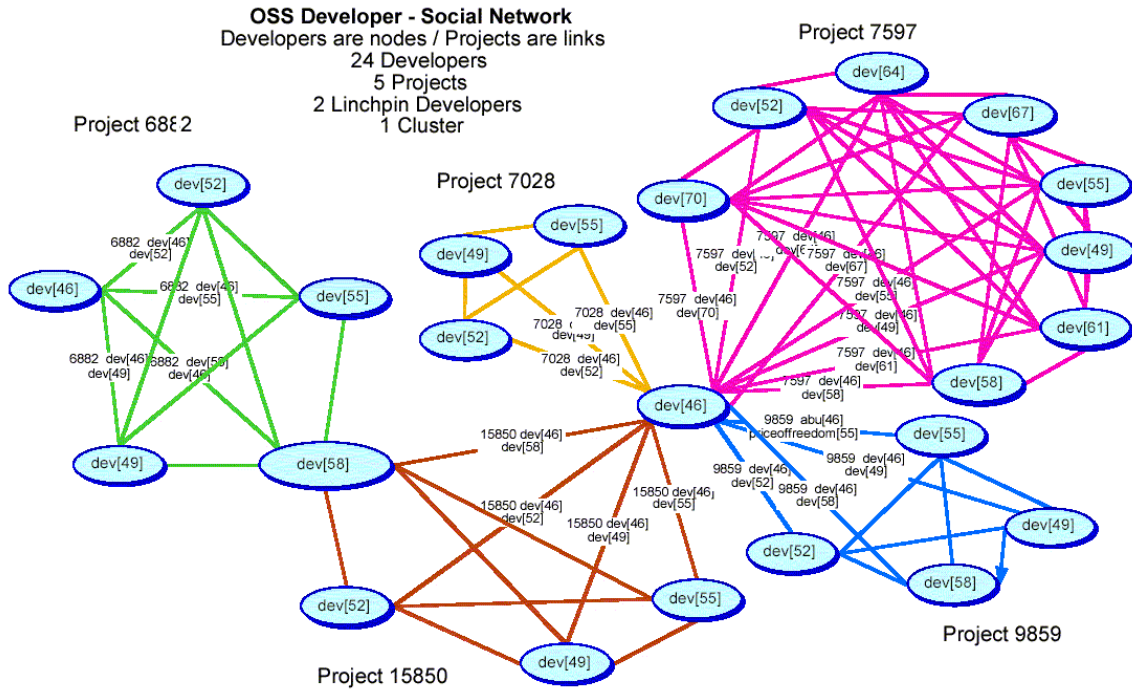


Figure 3. A social network that links 24 developers in five projects through two key developers into a larger F/OSS project community [cf. Madey 2004].

PlaneShift - A 3D Fantasy MMORPG - Mozilla

File Edit View Go Bookmarks Tools Window Help

Back Forward Reload Stop http://www.planeshift.it/helpus_recruit Search Print

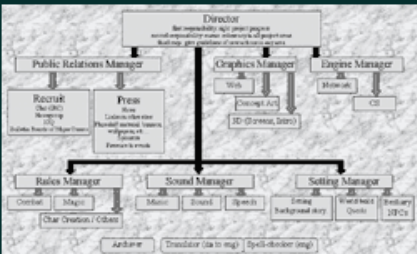
Join the Team!

Here are 10 reasons why you should dedicate some of your time (or your entire life, as you prefer) to PlaneShift:

1. PlaneShift is the first 3D MMORPG to be free for all players to play, as much as they want
2. Thousands of fans will see and enjoy what YOU made
3. It will not be your job, but your hobby, so you can really enjoy it and work in the areas you like best
4. The team is ever growing, with talented people from all over the world sharing the same dream
5. Unlike pro games, no commercial constraints such as money, time, or schedule can stop us. Only our own quality standards and talents determine how far we can go
6. We are a fair, friendly and happy team! With contributors from 11 countries currently, you can make new friends all around the world--all with talents and interests similar to your own
7. We have a unique license which we feel will ensure the success of the project and the integrity of the game we are creating
8. The core engine is released under GPL giving to it endless life and endless ability to improve.
9. You will have the chance to contribute to the fun of thousands of players
10. With our vision of free and open code and free play, Planeshift Will shake the gaming community in the next few years

Organization details

PlaneShift is a complex project and first of all it needs a good organization, for this reason we have divided the project in different departments. Each department has a leader that will ensure the progress and coordination of contributors.



Here you find an Organization Chart that explains which are the departments that you can choose for contributing. Click on it!

In the current state we are not a commercial organization so we can't provide a salary for contributors, members or leaders.

Please note that we accept only people with age of 18 or older. Our team is made of people from 18 to 33 actually.

Positions in the Team

LEADER:

To be a leader you must pass the approval of the director. Before that you will be considered a W.T.B. (Want To Be) Leader and only after proving that you have the right skills and dedication to the project you will officially become a leader.

There's one leader for each department and he can have also one co-leader helping in his job. He will ensure progress in his department completing the most important tasks in his area and will organize work of other members.

Exhibit 3. An example statement for how a F/OSS computer game development project seeks to organize and manage itself.

(Source: http://www.planeshift.it/helpus_recruit.html, March 2004).

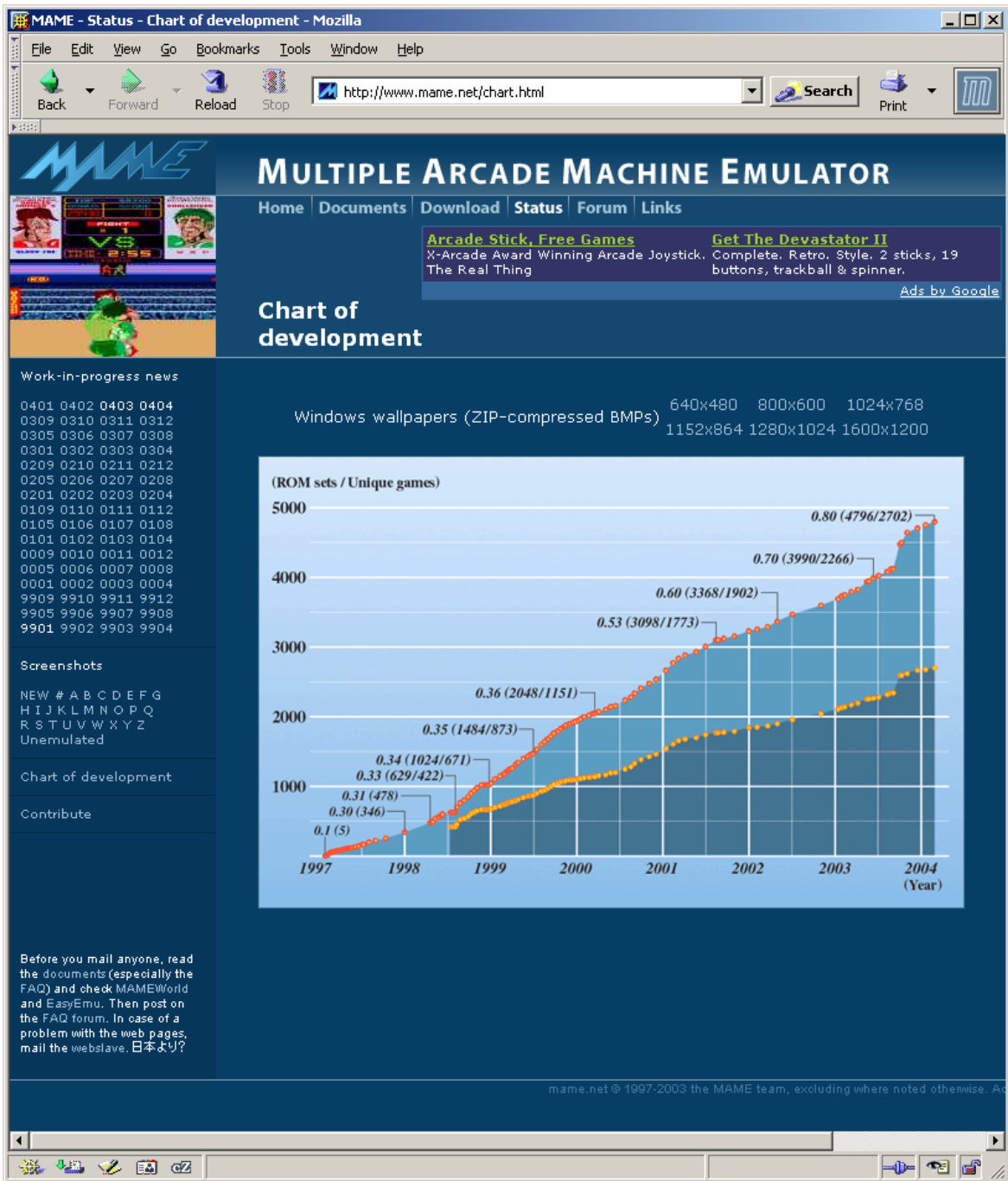


Exhibit 4. A graphic display depicting sustained growth in the number of vintage arcade ROM sets and games migrated into open source for use on contemporary computer platforms. (source: <http://www.mame.net/chart.html>, March 2004).