Understanding Requirements for Open Source Software

Walt Scacchi

Institute for Software Research, University of California-Irvine
Irvine, CA 92697-3425 USA
wscacchi@ics.uci.edu

Abstract. This study presents findings from an empirical study directed at understanding the roles, forms, and consequences arising in requirements for open source software (OSS) development efforts. Five open source software development communities are described, examined, and compared to help discover what differences may be observed. At least two dozen kinds of software informalisms are found to play a critical role in the elicitation, analysis, specification, validation, and management of requirements for developing OSS systems. Subsequently, understanding the roles these software informalisms take in a new formulation of the requirements development process for OSS is the focus of this study. This focus enables considering a reformulation of the requirements engineering process and its associated artifacts or (in)formalisms to better account for the requirements when developing OSS systems. Other findings identify how OSS requirements are decentralized across multiple informalisms, and to the need for advances in how to specify the capabilities of existing OSS systems.

Keywords: Open source software, Requirements process, Empirical studies, Decentralized software development, Artifacts.

1 Introduction

The focus in this paper is directed at understanding the requirements processes for open source software (OSS) development efforts, and how the development of these requirements differs from those traditional to software engineering and requirements engineering [1], [2], [3], [4], [5]. This study is about ongoing discovery, description, and abstraction of OSS development (OSSD) practices and artifacts in different settings across different communities. It is about expanding our notions of what requirements need to address to account for OSSD. Subsequently, these are used to understand what OSS communities are being examined, and what characteristics distinguish one community from another. This chapter also builds on, refines, and extends earlier study on this topic [6], [7], [8], [9], [10], as well as identifying implications for what requirements arise when developing different kinds of OSS systems.

This study reports on findings and results from an ongoing investigation of the socio-technical processes, work practices, and community forms found in OSSD [10], [11], [12]. The purpose of this multi-year investigation is to develop narrative, semi-structured (i.e., hypertextual), and formal computational models of these processes, practices, and community forms [8], [13]. This chapter presents a systematic narrative

K. Lyytinen et al. (Eds.): Design Requirements Workshop, LNBIP 14, pp. 467–494, 2009.

model that characterizes the processes through which the requirements for OSS systems are developed. The model compares in form, and presents an account of, how software requirements differ across traditional software engineering and OSS approaches. This model is descriptive and empirically grounded. The model is also comparative in that it attempts to characterize an open source requirements engineering process that transcends the practice in a particular project, or within a particular community. This comparative dimension is necessary to avoid premature generalizations about processes or practices associated with a particular OSS system or those that receive substantial attention in the news media (e.g., the GNU/Linux operating system). Such comparison also allows for system projects that may follow a different form or version of OSSD (e.g., those in the higher education computing community or networked computer game arena). Subsequently, the model is neither prescriptive nor proscriptive in that it does not characterize what should be or what might be done in order to develop OSS requirements, except in the concluding discussion, where such remarks are bracketed and qualified.

Comparative case studies of requirements or other software development processes are also important in that they can serve as foundation for the formalization of our findings and process models as a process meta-model [14]. Such a meta-model can be used to construct a predictive, testable, and incrementally refined theory of OSSD processes within or across communities or projects. A process meta-model is also used to configure, generate, or instantiate Web-based process modeling, prototyping, and enactment environments that enable modeled processes to be globally deployed and computationally supported (e.g., [8], [13], [15], [16]). This may be of most value to other academic research or commercial development organizations that seek to adopt "best practices" for OSSD processes that are well suited to their needs and situation. Therefore, the study and results presented in this report denote a new foundation on which computational models of OSS requirements processes may be developed, as well as their subsequent analysis and simulation (cf. [13], [17].

The study reported here entails the use of empirical field study methods [18] that follow conform to the principles for conducting and evaluating interpretive research design [19] as identified earlier [9].

2 Understanding OSS Development Across Different Communities

We assume there is no general model or globally accepted framework that defines how OSS is or should be developed. Subsequently, our starting point is to investigate OSS practices in different communities from an ethnographic perspective [16], [20], [21]. We have chosen five different communities to study. These are those centered about the development of software for networked computer games, Internet/Web infrastructure, bioinformatics and higher education computing. The following sections briefly introduce and characterize these OSS sub-domains. Along the way, example software systems or projects are *highlighted* or identified via external reference/citation, which can be consulted for further information or review.

2.1 Networked Computer Game Worlds

Participants in this community focus on the development and evolution of first person shooters (FPS) games (e.g., *Quake Arena*, *Unreal Tournament*), massive multiplayer online role-playing games (e.g., *World of Warcraft, Lineage, EveOnline, City of Heroes*), and others (e.g., *The Sims* (Electronic Arts), *Grand Theft Auto* (Rockstar Games)). Interest in networked computer games and gaming environments, as well as their single-user counterparts, have exploded in recent years as a major (now global) mode of entertainment, playful fun, and global computerization movement [22]. The release of *DOOM* [4], an early first-person action game, onto the Web in open source form[1] in the mid 1990's, began what is widely recognized the landmark event that launched the development and redistribution of computer game *mods* [9], [23].

Mods[2] are variants of proprietary (closed source) computer game engines that provide extension mechanisms like game scripting languages (e.g., UnrealScript for mod development with *Unreal* game engines) that can be used to modify and extend a game, and these extensions are licensed for distribution in an open source manner. Mods are created by small numbers of users who want and are able to modify games, compared to the huge numbers of players that enthusiastically use the games as provided. The scope of mods has expanded to now include new game types, game character models and skins (surface textures), levels (game play arenas or virtual worlds), and artificially intelligent game bots (in-game opponents).

Perhaps the most widely known and successful game mod is *Counter-Strike*, which is a total conversion of Valve Software's *Half-Life* computer game developed by two game programmers (Valve Software has since commercialized *CS* and many follow-on versions). Millions of copies of *CS* have been distributed, and millions of people have player *CS* over the Internet, according to http://counterstrikesource.net/. Other popular computer games that are frequent targets for modding include the *Quake, Unreal, Half-Life,* and *Crysis* game engines, *NeverWinter Nights* for role-playing games, motor racing simulation games (e.g., *GTR* series), and even the massively popular *World of Warcraft* (which only allows for modification of end-user interfaces). Thousands of game mods are distributed through game mod portals like MODDB.com. However, many successful game companies including Electronic Arts and Microsoft do not embrace nor encourage game modding, and do not provide end-user license agreements that allow game modding and redistribution.

---

[1] The end-user license agreement for games that allow for end-user created game mods often stipulate that the core game engine (or retail game software product) is protected as closed source, proprietary software that cannot be examined or redistributed, while any user created mod can only be redistributed as open source software that cannot be declared proprietary or sold outright, and must only be distributed in a manner where the retail game product must be owned by any end-user of a game mod. This has the effect of enabling a secondary market for retail game purchases by end-users or other game modders who are primarily interested in accessing, studying, playing, further modifying, and redistributing a game mod.

[2] For introductory background on computer game mods, see http://en.wikipedia.org/wiki/Mod_(computer_gaming).

2.2 Internet/Web Infrastructure

Participants in this community[3] focus on the development and evolution of systems like the *Apache* web server, *Mozilla/Firefox* Web browser[4], *GNOME* and *K Development Environment* (KDE) for end-user interfaces, the *Eclipse* and *NetBeans* interactive development environments for Java-based Web applications, and thousands of others. This community can be viewed as the one most typically considered in popular accounts of OSS projects. The GNU/Linux operating system environment is of course the largest, most complex, and most diverse sub-community within this arena, so much so that it merits separate treatment and examination. Many other Internet or Web infrastructure projects constitute recognizable communities or sub-communities of practice. The software systems that are the focus generally are not standalone end-user applications, but are often targeted at *system administrators* or *software developers as the targeted user base,* rather than the eventual end-users of the resulting systems. However, notable exceptions like Web browsers, news readers, instant messaging, and graphic image manipulation programs are growing in number within the end-user community

2.3 Bioinformatics

Participants in this community[5] focus on the development and evolution of software systems supporting research into bioinformatics and related computing-intensive biological research efforts. In contrast to the preceding two development oriented communities, OSS plays a significant role in scientific research communities. For example, when scientific findings or discoveries resulting from *in silico* experimentation or observations are reported[6], then members of the relevant scientific community want to be assured that the results are not the byproduct of some questionable software calculation or opaque processing trick. In scientific fields like astrophysics that

---

[3] The SourceForge web portal (http://www.sourceforge.net), the largest associated with the OSS community, currently stores information on more than 1,750K registered users and developers, along with nearly 200K OSSD projects (as of July 2008), with more than 10% of those projects indicating the availability of a mature, released, and actively supported software system. However, some of the most popular OSS projects have their own family of related projects, grouped within their own portals, such as for the Apache Foundation and Mozilla Foundation.

[4] It is reasonable to note that the two main software systems that enabled the World Wide Web, the *NCSA Mosaic* Web browser (and its descendants, like Netscape Navigator, Mozilla, Firefox, and variants like *K-Meleon*, *Konqueror*, *SeaMonkey*, and others), and the Apache Web server (originally know as *httpd*) were originally and still remain active OSSD projects.

[5] For information about OSS projects, activities, and events in this community, see http://www.bioinformatics.org, http://www.open-bio.org, and http://www.open-bio.org/wiki/Upcoming_BOSC_conference.

[6] For example, see [24]. The OSS processing pipelines for each sensor or mass spectrometer are mostly distinct and are maintained by different organizations. However, their outputs must be integrated, and the data source must be registered and oriented for synchronized alignment or overlay, then composed into a final representation (e.g., see [24]). Subsequently, many OSS programs may need to be brought into alignment for such a research method and observation,for a scientific discovery to be claimed and substantiated [25].

critically depend on software, open source is considered an essential precondition for research to proceed, and for scientific findings to be trusted and open to independent review and validation. Furthermore, as discoveries in the physics of deep space are made, this in turn often leads to modification or extension of the astronomical software in use in order to further explore and analyze newly observed phenomena, or to modify/add capabilities to how the remote sensing mechanisms operate.

2.4 Higher Education Computing

Participants in this community focus on the development and evolution of software supporting educational and administrative operations found in large universities or similar institutions. This community should not in general be associated with the activities of academic computer scientists nor of computer science departments, unless they specifically focus on higher education computing applications (which is uncommon). People who participate in this community generally develop software for academic teaching or administrative purposes in order to explore topics like course management (*Sakai, Moodle*), campuswide information systems/portals (*uPortal*), Web-based academic applications (*Fluid*), and university e-business systems [26] (for collecting student tuition, research grants administration, payroll, etc. -- *Kuali*). Projects in this community[7] are primarily organized and governed through multi-institution contracts, annual subscriptions, and dedicated staff assignments [27]. Furthermore, it appears that software developers in this community are often not the end-users of the software the develop, in contrast to most OSS projects. Accordingly, it may not be unreasonable to expect that OSS developed in this community should embody or demonstrate principles or best practices in administrative computing found in large public or non-profit enterprises, rather than commercial for-profit enterprises. This includes the practice of developing explicit software requirements specification documents prior to undertaking system development. Furthermore, much like the bioinformatics community, members of this community expect that when breakthrough technologies or innovations have been declared, such as in a refereed conference paper or publication in an educational computing journal, the opportunity exists for other community members to be able to access, review, or try out the software to assess and demonstrate its capabilities. Furthermore, there appears to be growing antagonism toward commercial software vendors (Blackboard Inc., PeopleSoft, Oracle) whose products target the higher education computing market (e.g., WebCT). However, higher education computing software is intended for routine production use by administrative end-users and others, and not research-grade "proof of concept" demonstration or prototype systems that are found in academic research laboratories.

---

[7] For information about OSS projects, events, and activities in this community, see http://www.sakaiproject.org, http://www.moodle.org, http://www.uportal.org, http://www.fluidproject. org, http://www.kuali.org, as well as EDUCAUSE (http://www. educause.edu/), a non-profit association focusing on current issues in information technology for higher education, including OSS development and OSS policy in academia.

2.5 Military Computing

Participants in this community[8] focus on the development and deployment of computing systems and applications that support secured military and combat operations. Although information on specific military systems may be limited, there are a small but growing number of sources of public information and OSS projects that support military and combat operations. Accordingly, it is becoming clear that the future of military computing, and the future acquisition of software-intensive, mission-critical systems for military or combat applications will increasingly rely on OSS [28], [29], [30], [31], [32], [33], [34], [35]. For example, the U.S. Army relies on tactical command and control systems hosted on Linux systems that support *Apache Tomcat* servers, *Jabber/XMPP* chat services, and *JBoss*-based Web services [30]. Other emerging applications are being developed for future combat systems, enterprise systems (the U.S. Department of Defense is the world's largest enterprise, with more than 1 million military and civilian employees), and various training systems, among others [33], [34], [35]. The development of software systems for developing simulators and game-based virtual worlds [36] are among those military software projects that operate publicly as a "traditional" OSS project that employs a GPL software license, while other projects operate as corporate source (i.e., OSS projects behind the corporate firewall) or community source projects, much like those identified for higher education computing [27].

2.6 Overall Cross-Community Characteristics

In contrast to efforts that draw attention to generally one (but sometimes many) open source development project(s) within a single community (e.g., [37], [38], [39], there is something to be gained by examining and comparing the communities, processes, and practices of OSSD in different communities. This may help clarify what observations may be specific to a given community (e.g., GNU/Linux projects), compared to those that span multiple, and mostly distinct communities. In this study, two of the communities are primarily oriented to develop software to support scholarly research or institutional administration (bioinformatics and higher education computing) with rather small user communities. In contrast, the other three communities are oriented primarily towards software development efforts that may replace/create commercially viable systems that are used by large end-user communities. Thus, there is a sample space that allows comparison of different kinds.

Each of these highlighted items point to the public availability of data that can be collected, analyzed, and re-represented within narrative ethnographies [40], [41], computational process models [13], [14], [17], or for quantitative studies [42], [43]. Significant examples of each kind of data have been collected and analyzed as part of this ongoing study. This paper includes a number of OSSD artifacts as data exhibits that empirically ground our analysis. These artifacts serve to document the social actions and technical practices that facilitate and constrain OSSD processes [7], [10],

[8] The primary source of information about OSS projects in the military comes from the cited references, rather than from publicly accessible Web sites. However, there are a few Military OSS projects accessible on the Web such as the Delta3D game engine at http://www.Delta3D.org, used to developed military training simulations.

[12], [44], [45]. Subsequently, we turn to review what requirements engineering is about, in order to establish how the process of developing OSS system requirements is similar or different than is common to traditional software engineering and information system development practices.

3 Informalisms for Describing OSS Requirements

The functional and non-functional requirements for OSS systems are elicited, analyzed, specified, validated, and managed through a variety of Web-based artifacts. These descriptive documents can be treated as software informalisms. *Software informalisms* [9] are the information resources and artifacts that participants use to describe, proscribe, or prescribe what's happening in a OSSD project. They are informal narrative resources codified in lean descriptions [cf. 46] that coalesce into *online document genres* (following [47], [48]) that are comparatively easy to use, and publicly accessible to those who want to join the project, or just browse around. In earlier work, Scacchi [9] demonstrates how software informalisms can take the place of formalisms, like "requirement specifications" or software design notations which are documentary artifacts seen as necessary to develop high quality software according to the software engineering community [1], [2], [3], [4], [5]. Yet these software informalisms often capture the detailed rationale, contextualized discourse, and debates for why changes were made in particular development activities, artifacts, or source code files. Nonetheless, the contents these informalisms embody require extensive review and comprehension by a developer before contributions can be made (cf. [49]). Finally, the choice to designate these descriptions as informalisms[9] is to draw a distinction between how the requirements of OSS systems are described, in contrast to the recommended use of formal, logic-based requirements notations ("formalisms") that are advocated in traditional approaches (cf. [1], [2], [3], [4], [5]).

In OSSD projects, software informalisms are the preferred scheme for describing or representing OSS requirements. There is no explicit objective or effort to treat these informalisms as "informal software requirements" that should be refined into formal requirements [3], [51], [52] within any of these communities. Accordingly, each of the available types of software requirements informalisms have been found in one or more of the five communities in this study. Along the way, we seek to identify some of the relations that link them together into more comprehensive stories, storylines, or intersecting story fragments that help convey as well as embody the requirements of an OSS system. Knowledge about who is doing what, where, when, why, and how is captured in different or multiple informalisms.

Two dozen types of software informalisms can be identified, and each has subtypes that can be identified. Those presented here are members of the set of software informalisms that are being used by different OSSD projects. Each OSSD project usually employs only a subset as its informal document ecology (cf. [47], [48]) that meets their interests or needs. There are no guidelines for which informalisms to use

[9] As Goguen [50] observes, formalisms are not limited to those based on a mathematical logic or state transition semantics, but can include descriptive schemes that are formed from structured or semi-structured narratives, such as those employed in Software Requirements Specifications documents.

for what, only observed practices that recur across OSSD projects. Thus it is premature and perhaps inappropriate to seek to further organize these informalisms into a classification or taxonomic scheme whose purpose is to prescribe when or where best to use one or another. Subsequently, they are presented here as an unordered list since to do so otherwise would transform this analysis from empirically ground, interpretative descriptions into untested, hypothetical prescriptions (cf. 19], [21]).

The most common informalisms used in OSSD projects include (i) communications and messages within project Email [46], (ii) threaded message discussion forums (see Exhibit 1), bulletin boards, or group blogs, (iii) news postings, and (iv) instant messaging or Internet relay chat. These enable developers and users to converse with one another in a lightweight, semi-structured manner, and now use of these tools is global across applications domains and cultures. As such, the discourse captured in these tools is a frequent source of OSS requirements. A handful of OSSD projects have found that summarizing these communications into (v) project digests [7] helps provide an overview of major development activities, problems, goals, or debates. These project digests represent multi-participant summaries that record and hyperlink the rationale accounting for focal project activities, development problems, current software quality status, and desired software functionality. Project digests (which sometimes are identified as "kernel cousins") record the discussion, debate, consideration of alternatives, code patches and initial operational/test results drawn from discussion forums, online chat transcripts, and related online artifacts (cf. [7]). Exhibit 1[10] provides an example of a project digest from the GNUe electronic business software project.

As OSS developers and user employ these informalisms, they have been found to also serve as carriers of technical beliefs and debates over desirable software features, social values (e.g., reciprocity, freedom of choice, freedom of expression), project community norms, as well as affiliation with the global OSS social movement [6], [10], [44].

Other common informalisms include (vi) scenarios of usage as linked Web pages or screenshots, (vii) how-to guides, (viii) to-do lists, (ix) Frequently Asked Questions, and other itemized lists, and (x) project Wikis, as well as (xi) traditional system documentation and (xii) external publications (e.g., [53], [54]). OSS (xiii) project property licenses (whether to assert collective ownership, transfer copyrights, insure "copyleft," or some other reciprocal agreement) are documents that also help to define what software or related project content are protected resources that can subsequently be shared, examined, modified, and redistributed.

Finally, (xiv) open software architecture diagrams, (xv) intra-application functionality realized via scripting languages like Perl and PhP, and the ability to either (xvi) incorporate externally developed software modules or "plug-ins", or (xvii) integrate software modules from other OSSD efforts, are all resources that are used informally, where or when needed according to the interests or actions of project participants.

All of the software informalisms are found or accessed from (xix) project related Web sites or portals. These Web environments are where most OSS software informalisms can be found, accessed, studied, modified, and redistributed [9].

[10] Each exhibit appears as a screenshot of a Web browsing session. It includes contextual information seen in a more complete display view, as is common in virtual ethnographic studies (cf. [10], [13], [40]).

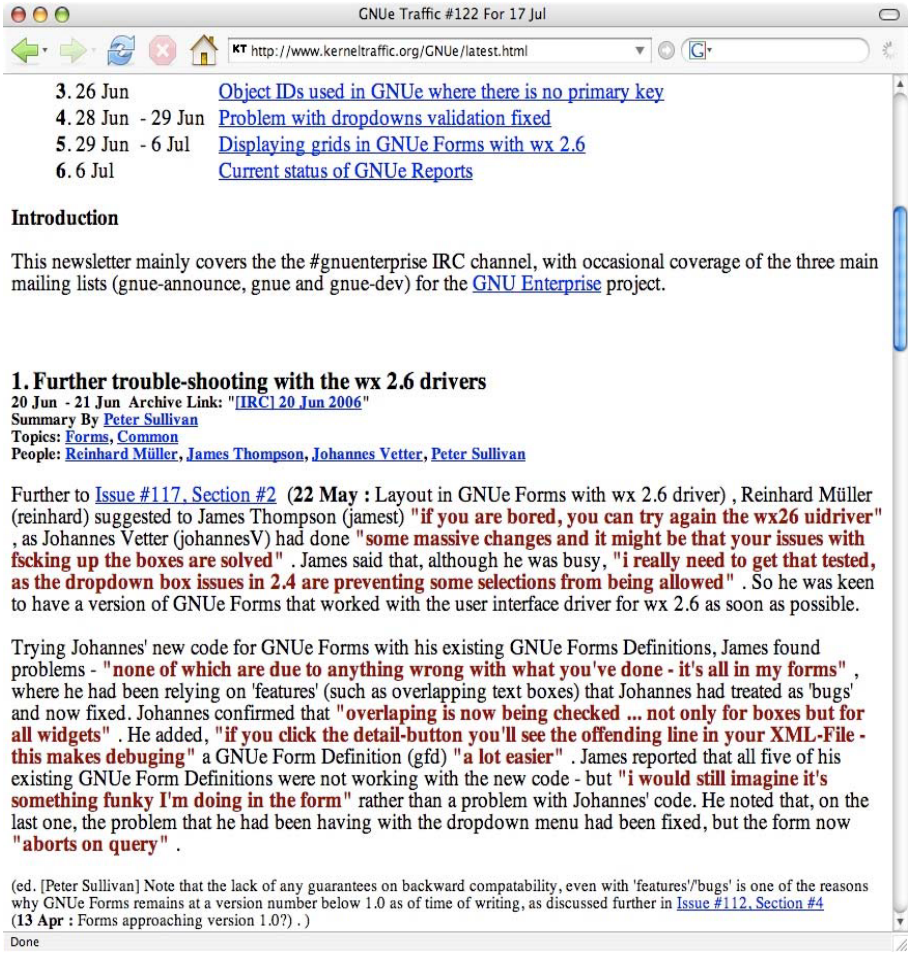

Exhibit 1. A project digest that summarizes multiple messages including those hyperlinked (indicated by highlighted and underlined text fonts) to their originating online sources. Source: http://www.kerneltraffic.org/GNUe/latest.html, July 2006.

A Web presence helps make visible the project's information infrastructure and the array of information resources that populate it. These include OSSD multi-project Web sites (e.g., SourgeForge.net, Savanah.org, Freshment.org, Tigris.org, Apache.org, Mozilla.org), community software Web sites (PhP-Nuke.org), and project-specific Web sites (e.g., www.GNUenterprise.org), as well as (xx) embedded project source code Webs (directories), (xxi) project repositories (CVS [53]), and (xxii) software bug reports and (xxiii) issue tracking data base like Bugzilla ([55], http://www.bugzilla.org/). Last, giving the growing global interest in online social networking, it not surprising to find increased attention to documenting various kinds of social gatherings and meetings using (xxiv) social media Web sites (e.g, YouTube,

Flickr, MySpace, etc.) where OSS developers, users, and interested others come together to discuss, debate, or work on OSS projects, and to use these online media to record, and publish photographs/videos that establish group identity and affiliation with different OSS projects.

Together, these two dozen types of software informalisms constitute a substantial yet continually evolving web of informal, semi-structured, or processable information resources that capture, organize, and distribute knowledge that embody the requirements for an OSSD project. This web results from the hyperlinking and cross-referencing that interrelate the contents of different informalisms together. Subsequently, these OSS informalisms are produced, used, consumed, or reused within and across OSSD projects. They also serve to act as both a distributed virtual repository of OSS project assets, as well as the continually adapted distributed knowledge base through which project participants evolve what they know about the software systems they develop and use.

Overall, it appears that none of these software informalisms would defy an effort to formalize them in some mathematical logic or analytically rigorous notation. Nonetheless, in the three of the five software communities examined in this study, there is no perceived requirement for such formalization (except for higher education computing and military computing), as the basis to improve the quality, usability, or cost-effectiveness of the OSS systems. If formalization of these documentary software informalisms has demonstrable benefit to members of these communities, beyond what they already realize from current practices, these benefits have yet to be articulated in the discourse that pervades each community. However, in contrast, the higher education and military communities do traditionally employ and develop formal requirements specification documents in order to coordinate and guide development of their respective "community source" software projects. Thus, we examine and compare these requirements development practices across all five communities so as to surface similarities, differences, and their consequences.

4 OSS Processes for Developing Requirements

In contrast to the world of classic software engineering, OSSD communities do not seem to readily adopt or practice modern software engineering or requirements engineering processes. Perhaps this is no surprise. If the history of software engineering were to reveal that one of the driving forces for capture and formalize software requirements was to support the needs of procurement and acquisition officials (i.e., not actual users of the resulting software) who want to be sure they know what some future software system is suppose to do once delivered, then requirements (documents) serve as the basis for software development contracts, delivery, and payment schedules. Software requirements are traditionally understood to serve a role in the development of proposed systems prior to their development [cf. 1], rather than for software systems that continuously emerge from networks of socio-technical interactions across a diverse ecosystem of users, developers, and other extant software systems [10], [11], [21]. However, OSS communities do develop software that is extremely valuable, generally reliable, often trustworthy, and readily used within its

associated user community. So, what processes or practices are being used to develop the requirements for OSS systems?

We have found many types of software requirements activities being employed within or across the five communities. However, what we have found in OSSD projects is different from common prescriptions for requirements engineering processes that seem to embraced in varying degrees by the higher education and military community source projects. The following subsections present six kinds of OSS requirements activities and associated artifacts that are compared with those traditional to software requirements engineering.

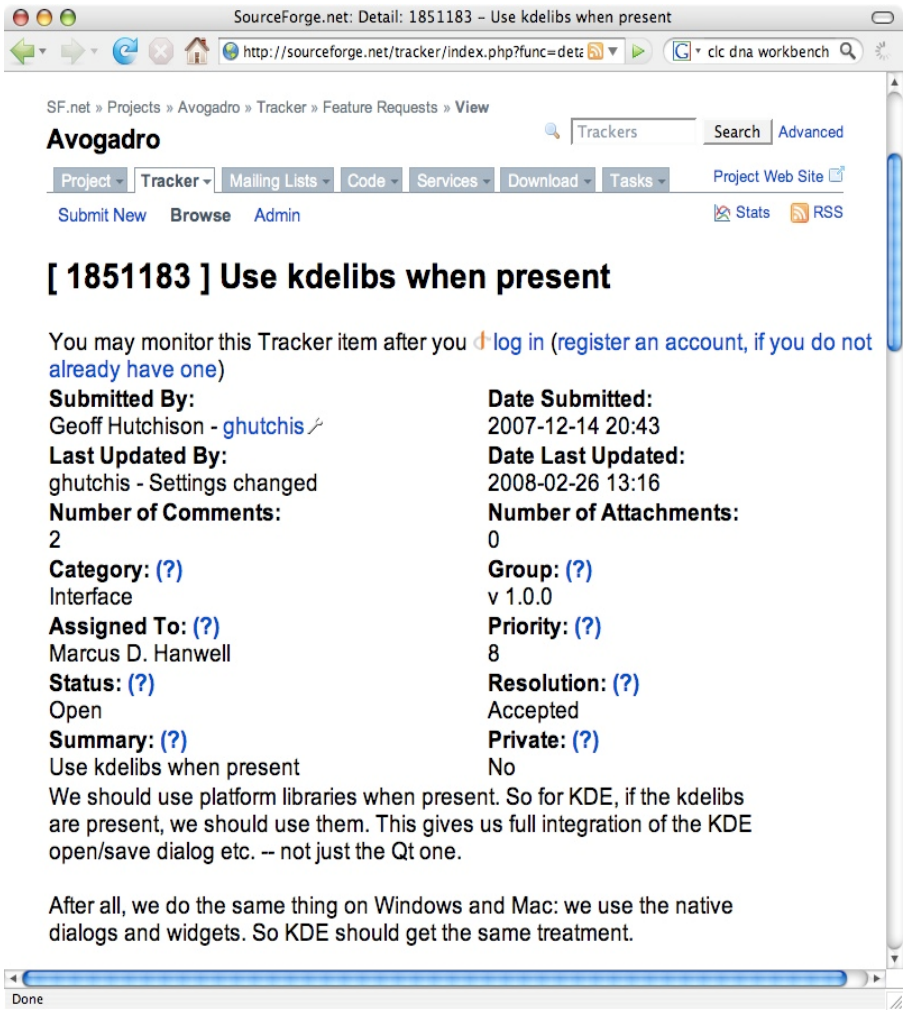

Exhibit 2. A sample of an asserted requirement to use the kdelibs platform libraries. Source: http://sourceforge.net/tracker/index.php?func=detail&aid=1851183&group_id=165310&atid=835080, June 2008.

4.1 Informal Post-Hoc Assertion of OSS Requirements vs. Requirements Elicitation

It appears that OSS requirements are articulated in a number of ways that are ultimately expressed, represented, or depicted on the Web. On closer examination, requirements for OSS can appear or be implied within an email message or within a discussion thread that is captured and/or posted on a Web site for open review, elaboration, refutation, or refinement. Consider the following example found on the Web site for the Avogardo molecular editor tool (http://avogadro.openmolecules.net) in the bioinformatics community. This example displayed in Exhibit 2 reveals the specification "We should use platform libraries when present. So for KDE, if the kdelibs are present, we should use them." As noted earlier, KDE is an Internet infrastructure community project.

These capabilities (appearing near the bottom of Exhibit 2) highlight implied requirements for the need or desirability of full integration of the Avogadro editor with the KDE functional command dialog system. These requirements are simply asserted without reference to other documents, standards, or end-user focus groups--they are requirements because some developers wanted these capabilities.

Perhaps it is more useful to define OSS requirements as *asserted system capabilities*. These capabilities are *post hoc* requirements characterizing a functional capability that has already been implemented, either in the system at hand, or by reference to some other related system that already exists. Based on observations and analyses presented here and elsewhere [6], [7], [8], [9], [10], [22], [45], it appears that concerned OSS developers assert and justify software system capabilities they support through their provision of the required coding effort to make these capabilities operational, or to modification some existing capability which may be perceived as limited or sometimes deficient. Senior members or core developers in the community then vote or agree through discussion to include the asserted capability into the system's feature set [56], or at least, not to object to their inclusion. The historical record their discourse and negotiation may be there, within the email or discussion forum archive, to document who required what, where, when, why, and how. However, once asserted, there is generally no further effort apparent to document, formalize, or substantiate such a capability as a system requirement. Asserted capabilities then become taken-for-granted requirements that are can be labeled or treated as *obvious* to those familiar with the system's development.

Another example reveals a different kind required OSSD capability. This case displayed in Exhibit 3, finds a requirements "mission" document that conveys a non-functional requirement for both community development and community software development in the bottom third of the exhibit. This can be read as a non-functional requirement for the system's developers to embrace community software development as the process to develop and evolve the ArgoUML system, rather than through a process that relies on the use of system models represented as UML diagrams.

Perhaps community software development, and by extension, community development, are recognized as socio-technical capabilities that are important to the development and success of this system. Regular practice of such capabilities may also be a method for improving system quality and reliability that can be compared to functional capabilities of existing software engineering tools and techniques that seem to

primarily focus on technical or formal analysis of software development artifacts as the primary way to improve quality and reliability.

The next example reveals yet another kind of elicitation found in the Internet/Web infrastructure community. In Exhibit 4, we see an overview of the MONO project. Here we see multiple statements for would-be software component/class owners to sign-up and commit to developing the required ideas, run-time, (object service) classes, and projects (cf. [45]). These are non-functional requirements for people to volunteer to participate in community software development, in a manner perhaps compatible with that portrayed in Exhibit 3.

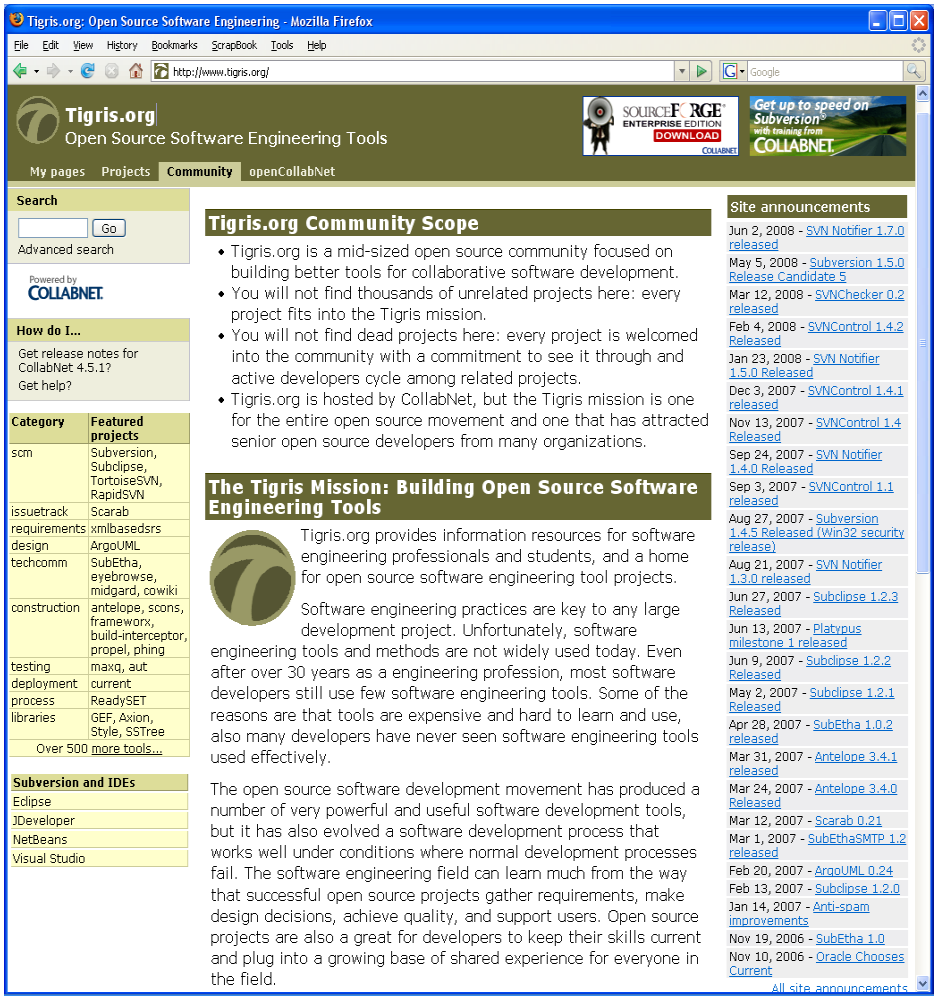

Exhibit 3. An OSS mission statement encouraging both the development of software for the community and development of the community. Source: http://www.tigris.org, June 2008.

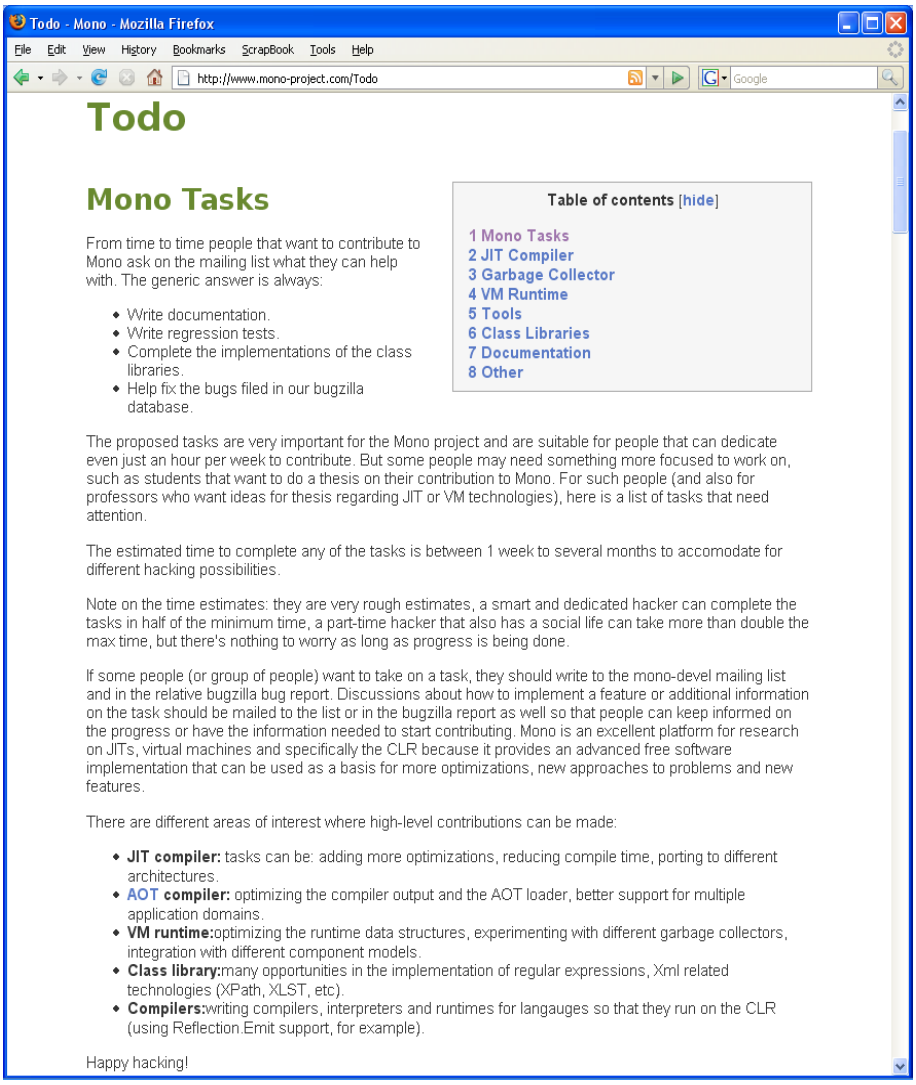

Exhibit 4. A non-functional requirement identifying a need for volunteers to become owners for yet to be developed software components whose functional requirements are still somewhat open and yet to be determined. Source: http://www.mono-project.com/Todo, June 2008.

The systems in Exhibit 3 must also be considered early in their overall development or maturity, because it calls for functional capabilities that are needed to help make the desired software functionality sufficiently complete for future usage. However, these yet "Todo" software implementation tasks signal to prospective OSS developers, who may want to join a project, as to what kinds of new software functionalities are desired, and thus telegraph a possible pathway for how to become a key

contributor within a large, established OSSD project [45] by developing a proposed software system component or function that some core developer desires.

Thus, in understanding how the capabilities and requirements of OSS systems are elicited, we find evidence for elicitation of volunteers to come forward to participate in community software development by proposing new software development projects, but only those that are compatible with the OSS engineering mission for the Tigris.org community.

We also observe the assertion of requirements that simply appear to exist without question or without trace to a point of origination, rather than somehow being elicited from stakeholders, customers, or prospective end-users of OSS systems. As previously noted, we have not yet found evidence or data to indicate the occurrence or documentation of a requirements elicitation effort arising in a traditional OSSD project. However, finding such evidence would not invalidate the other observations; instead, it would point to a need to broaden the scope of how software requirements are captured or recorded. For example, community source projects found in the higher education community seek to span OSSD practices with traditional software engineering practices, which results in hybrid software development and software requirements practices that do not seem to fully realize the practices (or benefits) of OSS engineering projects like those found at Tigris.org. Early experiences such a hybrid scheme suggest the successful software production may not directly follow [57].

4.2 Requirements Reading, Sense-Making, and Accountability vs. Requirements Analysis

Software requirements analysis helps identify what problems a software system is suppose to address and why, while requirements specifications identify a mapping of user problems to system based solutions. In OSSD, how does requirements analysis occur, and where and how are requirements specifications described? Though requirements analysis and specification are interrelated activities, rather than distinct stages, we first consider examining how OSS requirements are analyzed. In Exhibit 5 from the networked game community for the computer game *Unreal Tournament* (aka, UT3), it seems that game mod developers are encouraged to produce multi-version , continuously improving game mods, so that they can subsequently be recognized as professional game developers. Thus, OSS developers learn that achieving enhanced social status requires development of new software functions (mods) that improve across versions.

In seeking to analyze what is needed to more capably develop UT3 game mods, a game developer may seek additional information from other sources to determine how best to satisfy the challenge of developing a viable game mod. This in turn may lead one to discover and review secondary information sources, such as that shown in Exhibit 6. This exhibit points to still other Web-based information sources revealing both technical and social challenges that must be addressed to successfully develop a viable game mod.

Exhibit 5. An asserted capability (in the center) that invites would-be OSS game developers to make UT3 game mods, including improved versions, of whatever kind they require among the various types of available extensions so they may "get professional status," and possibly win money or other contest prizes. Source: http://www.ut3modding.com/, June 2008.

The notion that requirements for OSS system are, in practice, analyzed via the reading of technical accounts as narratives, together with making sense of how such readings are reconciled with one's prior knowledge, is not unique to the game modding software community. These same activities can and do occur in the other three communities. If one reviews the functional and non-functional requirements appearing in Exhibits 1-6, it is possible to observe that none of the descriptions appearing in these exhibits is self-contained. Instead, each requires the reader (e.g., a developer within the

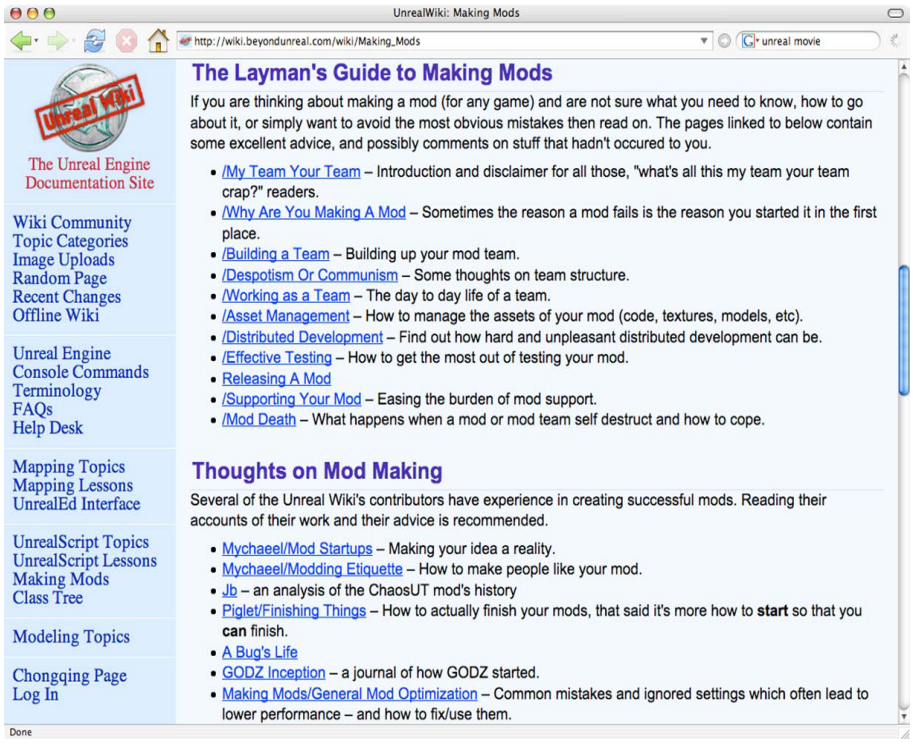

Exhibit 6. Understanding and analyzing what you need to know and do in order to develop a game mod. Source: http://wiki.beyondunreal.com/wiki/Making_Mods, May 2006.

community) to closely or casually read what is described, make sense of it, consult other materials or one's expertise, and trust that the description's author(s) are reliable and accountable in some manner for the OSS requirements that has been described [38], [50]. Analyzing OSS requirements entails little/no automated analysis, formal reasoning, or visual animation of software requirements specifications prior to the development of proposed software functionality (cf. [1], [5]). Instead, emphasis focuses on understanding what has already been accomplished in existing, operational system functionality, as well as what others have written and debated about it in different, project-specific informalisms. Subsequently, participants in these communities are able to understand what the functional and non-functional requirements are in ways that are sufficient to lead to the ongoing development of various kinds of OSS systems.

4.3 Continually Emerging Webs of Software Discourse vs. Requirements Specification and Modeling

If the requirements for OSS systems are asserted after code-based implementation rather than elicited prior to development of proposed system functionality, how are these OSS requirements specified or modeled? In traditional software development projects, the specification of requirements may be a deliverable required by contract. In most OSSD

projects, there are no such contractual obligations, and there are no requirements specification documents. In examining data from the five communities, of which Exhibits 1-6 are instances, it is becoming increasingly apparent that OSS capabilities can emerge both from the experiences of community participants using the software, as well as through iterative discussion and debate rendered in email and discussion forums. These communication messages in turn give rise to the development of narrative descriptions that more succinctly specify and condense into a web of discourse about the functional and non-functional requirements of an OSS system. This discourse is rendered in multiple, dispersed descriptions that can be found in email and discussion forum archives, on Web pages that populate community Web sites, and in other informal software descriptions that are posted, hyperlinked, or passively referenced through the assumed common knowledge that community participants expect their cohorts to possess. In this way, participating OSS developers and users collectively develop a deep, situated understanding of the capabilities they have realized and how unrealized needs must be argued for, negotiated, and otherwise be found to be obvious to the developers who see it in their self-interest to get them implemented.

In Exhibit 7 from the bioinformatics community, we see passing reference to the implied socio-technical requirement for bioinformatics scientists to program and orchestrate an e-science workflow to perform their research computing tasks. Such workflows are needed to realize a multi-step computational process that can be satisfied through an e-science tool/framework like Taverna (cf. [25], [58]). To comprehend and recognize what the requirements for bioinformatics workflows are in order to determine how to realize some bioinformatics data analysis or *in silico* experiment, community members who develop OSS for such applications will often be bioinformatics scientists (e.g., graduate students or researchers with Ph.D. degrees), and rarely would be simply a competent software engineering professional. Consequently, the bioinformatics scientists that develop software in this community do not need to recapitulate any software system requirement of the problem domain (e.g., microbiology). Instead, community members are already assumed to have mastery over such topics prior to software development, rather than encountering problems in their understanding of microbiology arising from technical problems in developing, operation, or functional enhancement of bioinformatics software. Subsequently, discussion and discourse focuses on how to use and extend the e-science workflow software in order to accomplish the scientific research to be realized through a computational workflow specification. Thus, a web of discourse can emerge about the functional requirement for specifying computational workflows that can be supported and documented by the software capabilities of an OSS workflow modeling tool like Traverna, rather than for specifying the functionality of the tool.

Thus, spanning the five communities and the seven exhibits, we begin to observe that the requirements for OSS are specified in webs of discourse that reference or link:

- email, bboard discussion threads, online chat transcripts or project digests,
- system mission statements,
- ideas about system functionality and the non-functional need for volunteer developers to implement the functionality,

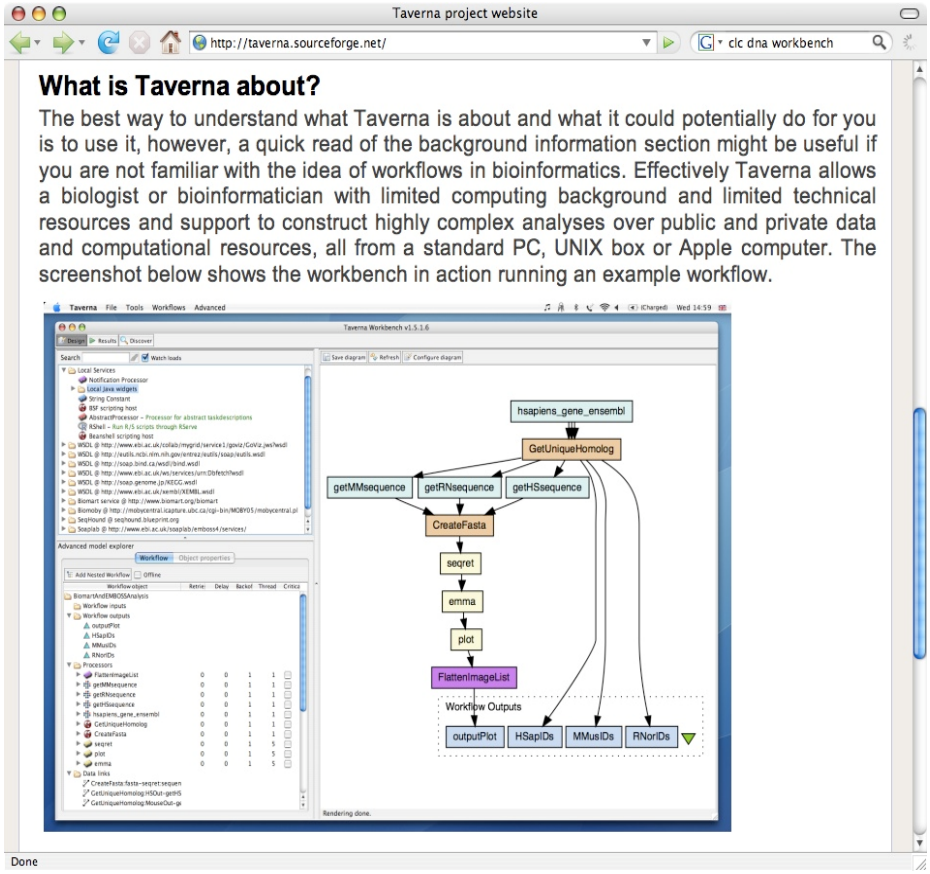

Exhibit 7. A description with embedded screenshot example of the Taverna tool framework for bioinformatics scientists suggesting why and how to develop workflows for computational processes needed to perform a complex data analysis or *in silico* research experiment [41]. Source http://taverna.sourceforce.net June 2008.

- promotional encouragement to specify and develop whatever functionality you need, which might also help you get a new job, and
- scholarly scientific research tools and publications that underscore how the requirements of bioinformatics software though complex, are understood without elaboration, since they rely on prior scientific knowledge and tradition of open scientific research.

Each of these modes of discourse, as well as their Web-based specification and dissemination, is a continually emerging source of OSS requirements from new contributions, new contributors or participants, new ideas, new career opportunities, and new research publications.

4.4 Condensing Discourse That Hardens and Concentrates System Functionality and Community Development vs. Requirements Validation

Software requirements are validated with respect to the software's implementation. The implemented system can be observed to demonstrate, exhibit, or be tested in operation to validate that its functional behavior conforms to its functional requirements. How are the software implementations to be validated against their requirements OSS requirements when they are not recorded in a formal Software Requirements Specifications document, nor are these requirements typically cast in a mathematical logic, algebraic, or state transition-based notational scheme?

In each of the five communities, it appears that the requirements for OSS are comingled with design, implementation, and testing descriptions and software artifacts, as well as with user manuals and usage artifacts (e.g., input data, program invocation scripts). Similarly, the requirements are spread across different kinds of artifacts including Web pages, sites, hypertext links, source code directories, threaded email transcripts, and more. In each community, requirements are routinely described, asserted, or implied informally. Yet it is possible to observe in threaded email discussions that community participants are able to comprehend and condense wide-ranging software requirements into succinct descriptions using lean media that pushes the context for their creation into the background.

Consider the next example found on the Web site for the KDE system (http://www.kde.org/), within the Internet/Web Infrastructure community. This example displayed in Exhibit 8 reveals asserted capabilities for the Qt3 subsystem within KDE, as well as displaying and documenting the part of the online discourse that justifies and explains the capabilities of the Qt3 subsystem in a manner that concentrates attention to processing features that the contributors find rationalizes the Qt3 requirements.

Goguen [50] suggests the metaphor of "concentrating and hardening of requirements" as a way to characterize how software requirements evolve into forms that are perceived as suitable for validation. His characterization seems to quite closely match what can be observed in the development of requirements for OSS. We find that requirements validation is a by-product, rather than an explicit goal, of how OSS requirements are constituted, described, discussed, cross-referenced, and hyperlinked to other informal descriptions of system and its implementations.

4.5 Global Access to OSS Webs vs. Communicating Requirements

One distinguishing feature of OSS associated with each of the five communities is that their requirements, informal as they are, are organized and typically stored in a persistent form that is globally accessible. This is true of community Web sites, site contents and hyperlinkage, source code directories, threaded email and other online discussion forums, descriptions of known bugs and desired system enhancements, records of multiple system versions, and more. Persistence, hypertext-style organization and linkage, and global access to OSS descriptions appear as conditions that do not receive much attention within the classic requirements engineering approaches, with few exceptions [51]. Yet, each of these conditions helps in the communication of OSS requirements. These conditions also contribute to the ability of community participants or outsiders looking in to trace the development and evolution of software requirements both within the software development descriptions, as well as across

community participants. This enables observers or developers to navigationally trace, for example, a web of different issues, positions, arguments, policy statements, and design rationales that support (e.g., see Exhibit 8) or challenge the viability of emerging software requirements (cf. [59], [60]).

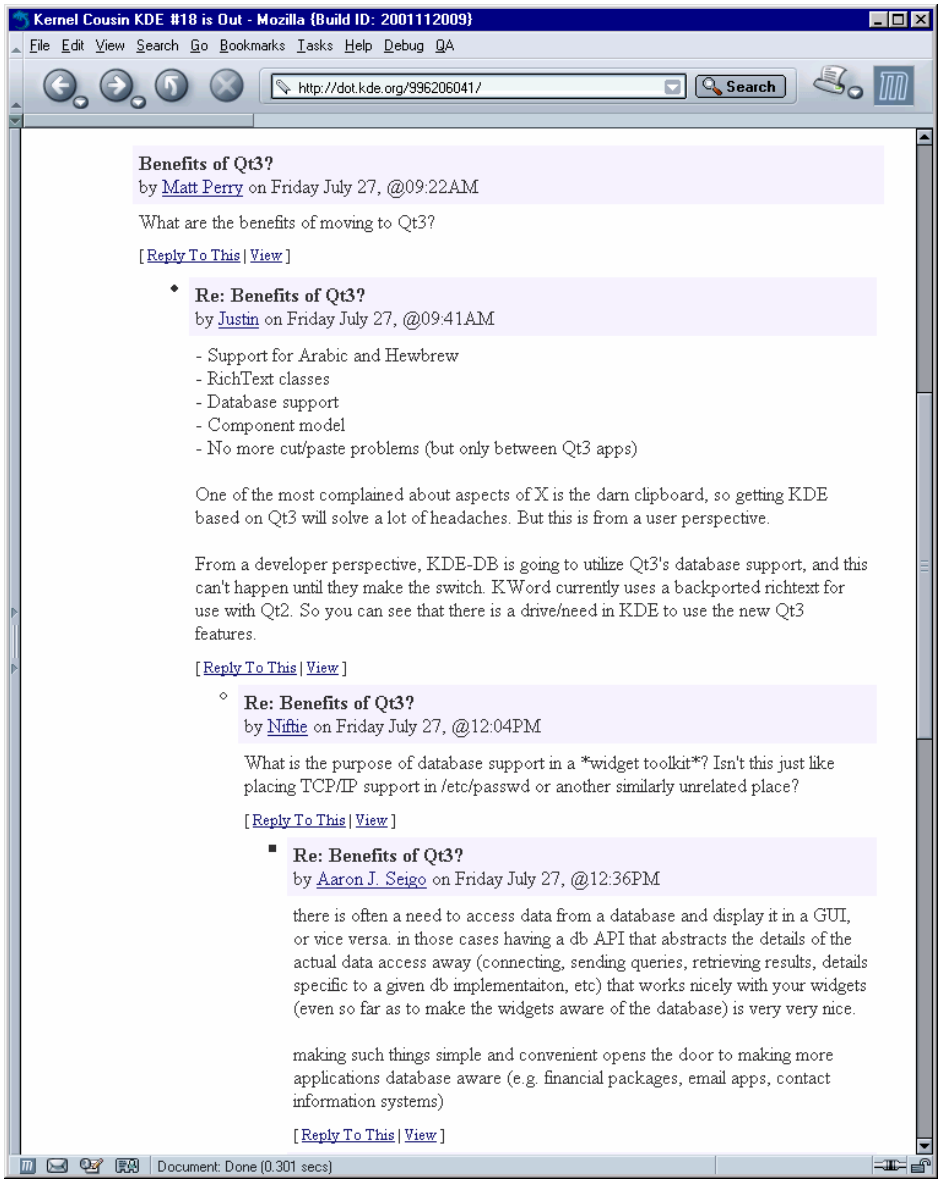

Exhibit 8. Asserted requirements and condensed justifications producing a hardened rationale for the KDE software subsystem Qt3 expressed through an online discourse. Source: http://dot.kde.org/996206041/, July 2001.

Each of the five communities also communicates community-oriented requirements. These non-functional requirements may seem similar to those for enterprise modeling [5], [61]. However, there are some differences, though they may be minor. First, each community is interested in sustaining and growing the community as a development enterprise (cf. [15]). Second, each community is interested in sustaining and growing the community's OSS artifacts, descriptions, and representations. Third, each community is interested in updating and evolving the community's information sharing Web sites. In recognition of these community requirements, it is not surprising to observe the emergence of commercial efforts (e.g., SourceForge and CollabNet) that offer community support systems that are intended to address these requirements, such as is used in projects like those in Tigris.org, or even the Avogadro project in the Bioinformatics community see (Exhibits 2 and 3).

4.6 Identifying a Common Foundation for the Development of OSS Requirements

Based on the data and analysis presented above, it is possible to begin to identify what items, practices, or capabilities may better characterize how requirements for OSS are developed and articulated. This centers around the preceding OSS requirements processes that enable the emergent creation, usage, and evolution of informal software descriptions as the vehicle for developing OSS requirements.

5 Understanding OSS Requirements

First, there is no single correct, right, or best way/method for constructing software system requirements. The requirements engineering approach long advocated by the software engineering and software requirements community does not account for the practice nor results of OSS system, project, or community requirements. OSS requirements (and subsequent system designs) are different. Thus, given the apparent success of sustained exponential growth for certain OSS systems [62], [63], and for the world-wide deployment of OSSD practices, it is safe to say that the ongoing development of OSS systems points to the continuous development, articulation, adaptation, and reinvention of their requirements (cf. [22]) as capabilities that emerge through socio-technical interactions between people, discursive artifacts, and the systems they use, rather than as needs to be captured before the proposed system comes into use.

Second, the traditional virtues of high-quality software system requirements, namely, their consistency, completeness, traceability, and internal correctness are not so valued in OSSD projects. OSSD projects focus attention and practice to other virtues that emphasize community development and participation, as well as other socio-technical concerns. Thus, as with the prior observation, OSS system requirements are different, and therefore may represent an alternative paradigm for how to develop robust systems that are open to both their developers and users. Nonetheless, there are many examples of the use of tools and techniques for articulating OSS requirements as well as for tracing or monitoring their development (cf. [61]).

Third, OSS developers are generally also end-users of the systems they develop. Thus, there is no "us-them" distinction regarding the roles of developers and end-users, as is commonly assumed in traditional system development practices. Because the developers are also end-users, communication gaps or misunderstandings often found between developers and end-users are typically minimized.

Fourth, OSS requirements tend to be distributed across space, time, people, and the artifacts that interlink them. OSS requirements are thus decentralized—that is, *decentralized requirements* that co-exist and co-evolve within different artifacts, online conversations, and repositories, as well as within the continually emerging interactions and collective actions of OSSD project participants and surrounding project social world. To be clear, decentralized requirements are not the same as the (centralized) requirements for decentralized systems or system development efforts. Traditional software engineering and system development projects assume that their requirements can be elicited, captured, analyzed, and managed as centrally controlled resources (or documentation artifacts) within a centralized administrative authority that adheres to contractual requirements and employs a centralized requirements artifact repository—that is, *centralized requirements*. Once again, OSS projects represent an alternative paradigm to that long advocated by software engineering and software requirements engineering community.

Last, given that OSS developers are frequently the source for the requirements they realize in hindsight (i.e., what they have successfully implemented and released denote what was required) rather than in foresight, perhaps it is better to characterize such software system requirements as instead "software system capabilities" (and not software development practices associated with capability maturity models). She or he who codes determines what the requirements will be based on what they have coded—the open source code frequently appears before there is some online discourse that specifies how and why it was done. OSS developers may simply tell others what was done, whether or not they discussed and debated it beforehand. They are generally under no contractual obligation to report and document software functionality prior to its coding and implementation. Subsequently, OSS capabilities embody requirements that have been found retrospectively to be both implementable and sustainable across releases. *Software capabilities specification*—specifying what the existing OSS system does—may therefore become a new engineering practice and methodology that can be investigated, modeled, supported, and refined. This in turn may then lead to principles for how best to specify software system capabilities.

6 Conclusions

The paper reports on a study that investigates, compares, and describes how the requirements engineering processes occurs in OSSD projects found in different communities. A number of conclusions can be drawn from the findings presented.

First, this study sought to discover and describe the practices and artifacts that characterize how the requirements for developing OSS systems. Perhaps the processes and artifacts that were described were obvious to the reader. This might be true for those scholars and students of software requirements engineering who have already participated in OSS projects, though advocates who have do not report on the

processes described here [37], [38], [39], [54]. For the majority of students who have not participated, it is disappointing to not find such descriptions, processes, or artifacts within the classic or contemporary literature on requirements engineering [1], [2], [3], [4], [5]. In contrast, this study sought to develop a baseline characterization of the how the requirements process for OSS occurs and the artifacts (and other mechanisms). Given such a baseline of the "as-is" process for OSS requirements engineering, it now becomes possible to juxtapose one or more "to-be" prescriptive models for the requirements engineering process, then begin to address what steps are needed to transform the as-is into the to-be [17]. Such a position provides a basis for further studies which seek to examine how to redesign OSS practices into those closer to advocated by classic or contemporary scholars of software requirements engineering. This would enable students or scholars of software requirements engineering, for example, to determine whether or not OSSD would benefit from more rigorous requirements elicitation, analysis, and management, and if so, how.

Second, this study reports on the centrality and importance of software informalisms to the development of OSS systems, projects, and communities. This result might be construed as an advocacy of the 'informal' over the 'formal' in how software system requirements are or should be developed and validated, though it is not so intended. Instead, attention to software informalisms used in OSS projects, without the need to coerce or transform them into more mathematically formal notations, raises the issue of what kinds of engineering virtues should be articulated to evaluate the quality, reliability, or feasibility of OSS system requirements so expressed. For example, traditional software requirements engineering advocates the need to assess requirements in terms of virtues like consistency, completeness, traceability, and correctness [1], [2], [3], [4], [5]. From the study presented here, it appears that OSS requirements artifacts might be assessed in terms of virtues like encouragement of community building; freedom of expression and multiplicity of expression; readability and ease of navigation; and implicit versus explicit structures for organizing, storing and sharing OSS requirements. "Low" measures of such virtues might potentially point to increased likelihood of a failure to develop a sustainable OSS system. Subsequently, improving the quality of such virtues for OSS requirements may benefit from tools that encourage community development; social interaction and communicative expression; software reading and comprehension; community hypertext portals and Web-based repositories. Nonetheless, resolving such issues is an appropriate subject for further study.

Overall, OSSD practices are giving rise to a new view of how complex software systems can be constructed, deployed, and evolved. OSSD does not adhere to the traditional engineering rationality found in the legacy of software engineering life cycle models or prescriptive standards. The development OSS system requirements is inherently and undeniably a complex web of socio-technical processes, development situations, and dynamically emerging development contexts [20], [21], [41], [50], [64]. In this way, the requirements for OSS systems continually emerge through a web of community narratives. These extended narratives embody discourse that is captured in persistent, globally accessible, OSS informalisms that serve as an organizational memory [65], hypertextual issue-based information system [7, 34], and a networked community environment for information sharing, communication, and social interaction [21], [44], [66], [67]. Consequently, ethnographic methods are

needed to elicit, analyze, validate, and communicate what these narratives are, what form they take, what practices and processes give them their form, and what research methods and principles are employed to examine them [5], [10], [13], [40], [41], [50], [64]. This report thus contributes a new study of this kind.

Acknowledgements

The research described in this report is supported by grants #0534771 from the U.S. National Science Foundation, as well as the Acquisition Research Program and the Center for the Edge Research Program, both at the Naval Postgraduate School. No endorsement implied. Chris Jensen, Thomas Alspaugh, John Noll, Margaret Elliott, and others at the Institute for Software Research are collaborators on the research project described in this paper.

References

1. Cheng, B.H.C., Atlee, J.M.: Research Directions in Requirements Engineering. In: Future of Software Engineering (FOSE 2007), pp. 285–303. IEEE Computer Society, Los Alamitos (2007)
2. Davis, A.M.: Software Requirements: Analysis and Specification. Prentice-Hall, Englewood Cliffs (1990)
3. Jackson, M.: Software Requirements & Specifications: Practice, Principles, and Prejudices. Addison-Wesley Pub. Co., Boston (1995)
4. Kushner, D.: Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture. Random House, New York (2003)
5. Nuseibeh, R., Easterbrook, S.: Requirements Engineering: A Roadmap. In: Finkelstein, A. (ed.) The Future of Software Engineering. ACM Press, New York (2000)
6. Elliott, M., Scacchi, W.: Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture. In: Koch, S. (ed.) Free/Open Source Software Development, pp. 152–172. IGI Publishing, Hershey (2005)
7. Elliott, M., Ackerman, M.S., Scacchi, W.: Knowledge Work Artifacts: Kernel Cousins for Free/Open Source Software Development. In: Proc. ACM Conf. Support Group Work (Group 2007), Sanibel Island, FL, pp. 177–186 (November 2007)
8. Jensen, C., Scacchi, W.: Process Modeling Across the Web Information Infrastructure. Software Process–Improvement and Practice 10(3), 255–272 (2005)
9. Scacchi, W.: Understanding the Requirements for Developing Open Source Software Systems. IEE Proceedings–Software 149(1), 24–39 (2002)
10. Scacchi, W.: Free/Open Source Software Development: Recent Research Results and Methods. In: Zelkowitz, M.V. (ed.) Advances in Computers, vol. 69, pp. 243–295 (2007)
11. Scacchi, W.: Socio-Technical Interaction Networks in Free/Open Source Software Development Processes. In: Acuña, S.T., Juristo, N. (eds.) Software Process Modeling, pp. 1–27. Springer Science+Business Media Inc., New York (2005)
12. Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K.: Understanding Free/Open Source Software Development Processes. Software Process–Improvement and Practice 11(2), 95–105 (2006)

13. Scacchi, W., Jensen, C., Noll, J., Elliott, M.: Multi-Modal Modeling, Analysis and Validation of Open Source Software Development Processes. Intern. J. Internet Technology and Web Engineering 1(3), 49–63 (2006)
14. Mi, P., Scacchi, W.: A Knowledge-based Environment for Modeling and Simulating Software Engineering Processes. IEEE Transactions on Knowledge and Data Engineering 2(3), 283–294 (1990)
15. Noll, J., Scacchi, W.: Supporting Software Development in Virtual Enterprises. J. Digital Information 1(4) (February 1999), http://jodi.ecs.soton.ac.uk/Articles/v01/i04/Noll/
16. Noll, J., Scacchi, W.: Specifying Process-Oriented Hypertext for Organizational Computing. J. Network and Computer Applications 24(1), 39–61 (2001)
17. Scacchi, W.: Understanding Software Process Redesign using Modeling, Analysis and Simulation. Software Process–Improvement and Practice 5(2/3), 183–195 (2000)
18. Zelkowitz, M.V., Wallace, D.: Experimental Models for Validating Technology. Computer 31(5), 23–31 (1998)
19. Klein, H., Myers, M.D.: A Set of Principles for Conducting and Evaluating Intrepretive Field Studies in Information Systems. MIS Quarterly 23(1), 67–94 (1999)
20. Ackerman, M.S., Atkinson, C.J.: Socio-Technical and Soft Approaches to Information Requirements Elicitation in the Post-Methodology Era. Requirements Engineering 5, 67–73 (2000)
21. Truex, D., Baskerville, R., Klein, H.: Growing Systems in an Emergent Organization. Communications ACM 42(8), 117–123 (1999)
22. Scacchi, W.: Free/Open Source Software Development Practices in the Computer Game Community. IEEE Software 21(1), 59–67 (2004)
23. Cleveland, C.: The Past, Present, and Future of PC Mod Development. Game Developer, 46–49 (February 2001)
24. Cagney, G., Amiri, S., Prewararadena, T., Lindo, M., Emili, A.: Silico proteome analysis to facilitate proteomic experiments using mass spectrometry. Proteome Science 1(5) (2003), doi:10.1186/1477-5956-1-5
25. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics J. 20(17), 3045–3054 (2004)
26. Scacchi, W.: Understanding the Development of Free E-Commerce/E-Business Software: A Resource-Based View. In: Sowe, S.K., Stamelos, I., Samoladas, I. (eds.) Emerging Free and Open Source Software Practices, pp. 170–190. IGI Publishing, Hershey (2007)
27. Wheeler, B.: Open Source 2010: Reflections on 2007, EDUCAUSE, pp. 49–67 (January/February 2007)
28. Bollinger, T.: Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense, The MITRE Corporation (January 2, 2001), http://www.terrybollinger.com/dodfoss/dodfoss_html/index.html
29. Guertin, N.: Naval Open Architecture: Open Architecture and Open Source in DOD, Open Source - Open Standards - Open Architecture, Association for Enterprise Integration Symposium, Arlington VA (March 14, 2007)
30. Justice, N.: Open Source Software Challenge: Delivering Warfighter Value, Open Source - Open Standards - Open Architecture, Association for Enterprise Integration Symposium, Arlington VA (March 14, 2007)
31. Riechers, C.: The Role of Open Technology in Improving USAF Software Acquisition, Open Source - Open Standards - Open Architecture, Association for Enterprise Integration Symposium, Arlington VA (March 14, 2007)

32. Scacchi, W., Alspaugh, T.: Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense. In: Proc. 5th Annual Acquisition Research Symposium, Naval Postgraduate School, Monterey, CA (2008)
33. Starrett, E.: Software Acquisition in the Army. Crosstalk: The Journal of Defense Software Engineering, 4–8 (May 2007)
34. Weathersby, J.M.: Open Source Software and the Long Road to Sustainability within the U.S. DoD IT System. The DoD Software Tech. News 10(2), 20–23 (2007)
35. Wheeler, D.A.: Open Source Software (OSS) in U.S. Government Acquisitions. The DoD Software Tech. News 10(2), 7–13 (2007)
36. McDowell, P., Darken, R., Sullivan, J., Johnson, E.: Delta3D: A Complete Open Source Game and Simulation Engine for Building Military Training Systems. J. Defense Modeling and Simulation: Applications, Methodology, Technology 3(3), 143–154 (2006)
37. DiBona, C., Ockman, S., Stone, M.: Open Sources: Voices from the Open Source Revolution. O'Reilly Press, Sebastopol (1999)
38. Pavlicek, R.: Embracing Insanity: Open Source Software Development. SAMS Publishing, Indianapolis (2000)
39. Raymond, E.: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly and Associates, Sebastopol (2001)
40. Hine, C.: Virtual Ethnography. SAGE Publishers, London (2000)
41. Kling, R., Scacchi, W.: The Web of Computing: Computer technology as social organization. In: Yovits, M. (ed.) Advances in Computers, vol. 21, pp. 3–90. Academic Press, New York (1982)
42. Howison, J., Conklin, M., Crowston, K.: Flossmole: A collaborative repository for floss research, data, and analysis. Intern. J. Information Technology and Web Engineering 1(3), 17–26 (2006)
43. Madey, G., Freeh, V., Tynan, R.: Modeling the F/OSS Community: A Quantitative Investigation. In: Koch, S. (ed.) Free/Open Source Software Development, pp. 203–221. Idea Group Publishing, Hershey (2005)
44. Elliott, M., Scacchi, W.: Mobilization of Software Developers: The Free Software Movement. Information, Technology and People 21(1), 4–33 (2008)
45. Jensen, C., Scacchi, W.: Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study. In: Proc. 29th. Intern. Conf. Software Engineering, Minneapolis, MN, pp. 364–374. ACM Press, New York (2007)
46. Yamaguchi, Y., Yokozawa, M., Shinohara, T., Ishida, T.: Collaboration with Lean Media: How Open-Source Software Succeeds. In: Proceedings of the Conference on Computer Supported Cooperative Work (CSCW 2000), Philadelphia, PA, pp. 329–338. ACM Press, New York (2000)
47. Kwansik, B., Crowston, K.: Introduction to the special issue: Genres of digital documents. Information, Technology and People 18(2) (2005)
48. Spinuzzi, C.: Tracing Genres through Organizations: A Sociocultural Approach to Information Design. MIT Press, Cambridge (2003)
49. Lanzara, G.F., Morner, M.: Artifacts rule! How organizing happens in open software projects. In: Czarniawska, B., Hernes, T. (eds.) Actor Network Theory and Organizing. Copenhagen Business School Press, Copenhagen (2005)
50. Goguen, J.A.: Formality and Informality in Requirements Engineering (Keynote Address). In: Proc. 4th. Intern. Conf. Requirements Engineering, pp. 102–108. IEEE Computer Society, Los Alamitos (1996)

51. Cybulski, J.L., Reed, K.: Computer-Assisted Analysis and Refinement of Informal Software Requirements Documents. In: Proceedings Asia-Pacific Software Engineering Conference (APSEC 1998), Taipei, Taiwan, R.O.C., pp. 128–135 (December 1998)
52. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. John Wiley and Sons, Inc., New York (1998)
53. Fogel, K.: Open Source Development with CVS. Coriolis Press, Scottsdale (1999)
54. Fogel, K.: Producing Open Source Software: How to Run a Successful Free Software Project. O'Reilly Press, Sebastopol (2005)
55. Ripoche, G., Gasser, L.: Scalable Automatic Extraction of Process Models for Understanding F/OSS Bug Repair. In: Proc. 16th Intern. Conf. Software Engineering & its Applications (ICSSEA 2003), Paris, France (December 2003)
56. Fielding, R.T.: Shared Leadership in the Apache Project. Communications of the ACM 42(4), 42–43 (1999)
57. Rosenberg, S.: Dreaming in Code: Two Dozen Programmers, Three years, 4732 Bugs, and One Quest for Transcendent Software. Crown Publishers, New York (2007)
58. Howison, J., Wiggins, A., Crowston, K.: eResearch workflows for studying free and open source software development. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) IFIP International Federation for Information Processing, Milan, IT. Open Source Development, Communities, and Quality, vol. 275 (2008)
59. Conklin, J., Begeman, M.L.: gIBIS: A Hypertext Tool for Effective Policy Discussion. ACM Transactions Office Information Systems 6(4), 303–331 (1988)
60. Lee, J.: SIBYL: a tool for managing group design rationale. In: Proc. Conf. Computer-Supported Cooperative Work (CSCW 1990), Los Angeles, CA, pp. 79–92. ACM Press, New York (1990)
61. Robinson, W.: A Requirements Monitoring Framework for Enterprise Systems. Requirements Engineering 11(1), 17–41 (2006)
62. Deshpande, A., Riehle, D.: The Total Growth of Open Source Software. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) IFIP International Federation for Information Processing, Open Source Development, Communities, and Quality, Milan, IT, vol. 275 (2008)
63. Scacchi, W.: Understanding Free/Open Source Software Evolution. In: Madhavji, N.H., Ramil, J.F., Perry, D. (eds.) Software Evolution and Feedback: Theory and Practice, pp. 181–206. John Wiley and Sons Inc., New York (2006)
64. Viller, S., Sommerville, I.: Ethnographically informed analysis for software engineers. Int. J. Human-Computer Studies 53, 169–196 (2000)
65. Ackerman, M.S., Halverson, C.A.: Reexamining Organizational Memory. Communications of the ACM 43(1), 59–64 (2000)
66. Kim, A.J.: Community-Building on the Web: Secret Strategies for Successful Online Communities. Peachpit Press (2000)
67. Smith, M., Kollock, P. (eds.): Communities in Cyberspace. Routledge, London (1999)