Understanding Requirements for Open Source Software

Walt Scacchi

Institute for Software Research

University of California, Irvine

Irvine, CA 92697-3425 USA

http://www.ics.uci.edu/~wscacchi

wscacchi@ics.uci.edu

June 2008

Abstract

This study presents findings from an empirical study directed at understanding the roles, forms, and consequences arising in requirements within open source software development efforts. Four open source software development communities are described, examined, and compared to help discover what these differences may be. 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 open source software systems. Subsequently, understanding the roles these software informalisms take in a new formulation of the requirements development process for open source software 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 for developing open source software systems.

1. Overview

The focus in this paper is directed at understanding the requirements processes for open source software development efforts, and how the development of these requirements differs from those traditional to software engineering and requirements engineering [Davis 1990 Jackson 1995, Kotonya 1998, Nuseibeh 2000]. This study is about ongoing discovery, description, and abstraction of open source software development practices and artifacts in different settings across different communities. It is about expanding our notions of what requirements need to address to account for open source software development. Subsequently, these are used to understand what open source software communities are being examined, and what characteristics distinguish one community from another. This chapter also builds on, refines, and extends an earlier study on this topic [Scacchi 2002], as well as identifying implications for what requirements arise when developing different kinds of open source software systems.

This study reports on findings and results from an ongoing investigation of the socio-technical processes, work practices, and community forms found in open source software development. 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 [Scaccchi, *et al.,* 2006]. This chapter presents a systematic narrative model that characterizes the processes through which the requirements for open source software systems are developed. The model compares in form, and presents a contrasting account of, how software requirements differ across traditional software engineering and open source 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 open source software 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 open source software development (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 open source software 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 [Mi 1990]. Such a meta-model can be used to construct a predictive, testable, and incrementally refined theory of open source software development processes within or across communities or projects. A process meta-model is also used to configure, generate, or instantiate Web-based process modeling, prototyping, and enactment environments that enable modeled processes to be globally deployed and computationally supported [e.g., Noll 1999, Noll 2001, Jensen 2005]. This may be of most value to other academic research or commercial development organizations that seek to adopt "best practices" for open source software development 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 open source software requirements

processes may be developed, as well as their subsequent analysis and simulation [cf. Scacchi 2000, Scacchi, *et al*., 2006].

The study reported here entails the use of empirical field study methods [Zelkowitz 1998] that follow conform to the principles for conducting and evaluating interpretive research design [Klein 1999] as identified earlier [Scacchi 2002].

2. Understanding open source software development across different communities

We assume there is no general model or globally accepted framework that defines how open source software is or should be developed. Subsequently, our starting point is to investigate open source software practices in different communities from an ethnographic perspective [Atkinson 2000 Nuseibeh 2000, Viller 2000].

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.

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 [Scacchi 2004, Scacchi 2008]. The release of DOOM, an early first-person action game, onto the Web in open source form in the mid 1990's, began what is widely recognized the landmark event that launched the development and redistribution of computer game *mods* [Cleveland 2001, Scacchi 2002]. Mods are variants of proprietary (closed source) computer game engines that provide extension mechanisms like game scripting languages 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).

2.2 Internet/Web infrastructure

Participants in this community focus on the development and evolution of systems like the Apache web server, Mozilla Firefox Web browser[1], 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[2]. This community can be viewed as the one most typically considered in popular accounts of open source software projects. The GNU/Linux operating system environment is of course the largest, most complex, and most diverse sub-community within this

[1] 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 off-shoots like K-Meleon, Konqueror, SeaMonkey, and others), and the Apache Web server (originally know as "httpd") were originally and still remain active open source software development projects.

[2] The SourceForge community web portal (http://www.sourceforge.net) currently stores information on more than 1,750K registered users and developers, along with nearly 200K open source software development projects, with more than 10% of those projects indicating the availability of a mature, released, and actively supported software system.

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 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, open source software plays a significant role in scientific research communities. For example, when scientific findings or discoveries resulting from remotely sensed observations are reported[3], 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 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

[3] For example, see http://XXXXX. The open source software processing pipelines for each sensor 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, as shown on the cited Web page. There are dozens of open source software programs that must be brought into alignment for such an image to be produced, and for such a scientific discovery to be claimed and substantiated [**xxxx**].

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 (SakaiProject.org), campuswide information systems/portals (uPortal.org), and university financial systems (for collecting student tuition, research grants administration, payroll, etc. -- Kuali.org). Projects in this community are primarily organized and governed through multi-institution contracts, annual subscriptions, and dedicated staff assignments [Wheeler 2007a]. Furthermore, it appears that software developers in this community are often not the end-users of the software the develop, in contrast to most FOSS projects. Accordingly, it may not be unreasonable to expect that open source software 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 whose products target the higher education computing market (e.g., WebCT, PeopleSoft/Oracle). However, it is often unacceptable to find that higher education computing software constitutes nothing more than a research-grade "proof of concept" demonstration or prototype system, not intended for routine or production use by end-users.

2.5 Military computing

Participants in this community focus on the development and deployment of computing systems and applications that support 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 open source software projects that support military and combat operations, 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 open source software [Guertin 2007, Justice 2007, Reichers 2007, Scacchi and Alspaugh 2008 , Starrett 2007, Weathersby 2007, Wheeler 2007b]. For example, it is now known that combat operations in the Iraq war directed by the U.S. Army rely on tactical command and control systems hosted on thousands of Linux systems, along with this deployment representing largest system support contract for Red Hat Linux [Justice 2007]. 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 [Starrett 2007, Weathersby 2007, Wheeler 2007b]. The development of software systems for developing simulators

and game-based virtual worlds [McDowell 2006] are among those military software projects that operate publicly as a "traditional" FOSS project that employs a GPL software license, while other projects operate as corporate source (i.e., FOSS projects behind the corporate firewall) or community source projects, much like those identified for higher education computing [Wheeler 2007a].

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., DiBona 1999, Raymond 2001], there is something to be gained by examining and comparing the communities, processes, and practices of open source software development 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 [Hine 2000, Kling 1982], computational process models [Mi 1990, Scacchi 2000, Scacchi, *et al*. 2006], or for quantitative studies [Madey 2005, Howison 2006]. Significant examples of each kind of data have been collected and analyzed as part of this ongoing study. This paper includes a number of examples that serve as this data.

Subsequently, we turn to review what requirements engineering is about, in order to establish a baseline of comparison for whether what we observe with the development of open source software system requirements is similar or different, and if so how.

3. Informalisms for describing open source software requirements

The functional and non-functional requirements for open source software systems are elicited, analyzed, specified, validated, and managed through a variety of Web-based descriptions. These descriptions can be treated as software informalisms. *Software informalisms* [Scacchi 2002] are the information resources and artifacts that participants use to describe, proscribe, or prescribe what's happening in a FOSSD project. They are informal narrative resources codified in lean descriptions [cf. Yamaguchi 2000] that coalesce into *online document genres* (following Kwansik and Crowston 2005, Spinuzzi 2003) that are comparatively easy to use, and publicly accessible to those who want to join the project, or just browse around. Subsequently, Scacchi [2002] demonstrates how software informalisms can take the place of formalisms, like "requirement specifications" or software design notations which are seen as necessary to develop high quality software according to the software engineering community [cf. Sommerville 2004]. Yet these software informalisms often capture the detailed rationale 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. Lanzara and Morner 2005]. Finally, the choice to designate these descriptions as informalisms[4] is to draw a distinction between how the requirements of open source software systems are described, in contrast to the

[4] As Goguen [2000] 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.

recommended use of formal, logic-based requirements notations ("formalisms") that are advocated in traditional approaches [cf. Davis 1990, Jackson 1995, Kotonya 1998, Nuseibeh 2000].

In OSSD projects, software informalisms are the preferred scheme for describing or representing open source software requirements. There is no explicit objective or effort to treat these informalisms as "informal software requirements" that should be refined into formal requirements [Cybulski 1998 Jackson 1995, Kotonya 1998] within any of these communities. Accordingly, each of the available types of software requirements informalisms t have been found in one or more of the four 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 open source software system.

At least two dozen types of software informalisms can be identified, and each has sub-types that can be identified as follows.

The most common informalisms used in FOSSD projects include (i) communications and messages within project Email [Yamaguchi 2000], (ii) threaded message discussion forums (see Exhibit 1), bulletin boards, or group blogs, (iii) news postings, (iv) project digests, and (v) instant messaging or Internet relay chat [Elliott 2007]. As FOSS 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 FOSS social movement [Elliott 2005, 2008].

Other common informalisms also include (vi) scenarios of usage as linked Web pages, (vii) how-to guides, (viii) to-do lists, (ix) FAQs, and other itemized lists, and (x) project Wikis, as well as (xi) traditional system documentation and (xii) external publications [e.g., Fogel 1999, 2005]. FOSS (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 where most FOSS software informalisms can be found, accessed, studied, modified, and redistributed [Scacchi 2002].

A Web presence helps make visible the project's information infrastructure and the array of information resources that populate it. These include FOSSD 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 [Fogel 1999]), and (xxii) software bug reports and (xxiii) issue tracking data base like Bugzilla

[Ripoche 2003, 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 FOSS developers, users, and interested others come together to discuss, debate, or work on FOSS projects, and to use these online media to record, and publish photographs/videos that establish group identity and affiliation with different FOSS projects.

Together, these two dozen types of software informalisms constitute a substantial yet continually evolving web of informal, semi-structured, or processable information resources. This web results from the hyperlinking and cross-referencing that interrelate the contents of different informalisms together. Subsequently, these FOSS informalisms are produced, used, consumed, or reused within and across FOSS development projects. They also serve to act as both a distributed virtual repository of FOSS 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 four software communities examined in this study, there is no perceived requirement for such formalization, nor no unrecognized opportunity to somehow improve the quality, usability, or cost-effectiveness of the open source software systems, that has been missed. If formalization of these software benefits 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.

4. Open source software processes for developing requirements

In contrast to the world of classic software engineering, open source software development communities do not seem to readily adopt or practice modern software engineering or requirements engineering processes. Perhaps this is no surprise. However, these 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 open source software systems?

We have found many types of software requirements activities being employed within or across the four communities. However, what we have found is different from common prescriptions for requirements engineering processes.

4.1 Requirements elicitation vs. assertion of open source software requirements

It appears that open source software requirements are articulated in a number of ways that are ultimately expressed, represented, or depicted on the Web. On closer examination, requirements for open source software 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 KDE system (http://www.kde.org/),

within the Internet/Web Infrastructure community. This example displayed in Exhibit 1[5] reveals asserted capabilities for the Qt3 subsystem within KDE.

These capabilities (identified in the exhibit as the "Re: Benefits of Qt3?" discussion thread) highlight implied requirements for multi-language character sets (Arabic and Hebrew, as well as English), database support ("…there is often need to access data from a database and display it in a GUI, or vice versa…"), and others. These requirements are simply asserted without reference to other documents, sources, standards, or joint application development (JAD) focus groups--they are requirements because some developers wanted these capabilities.

[5] Each exhibit appears as a screenshot of a Web browsing session. It includes contextual information, following the second research principle, thus requiring and benefiting from a more complete display view.

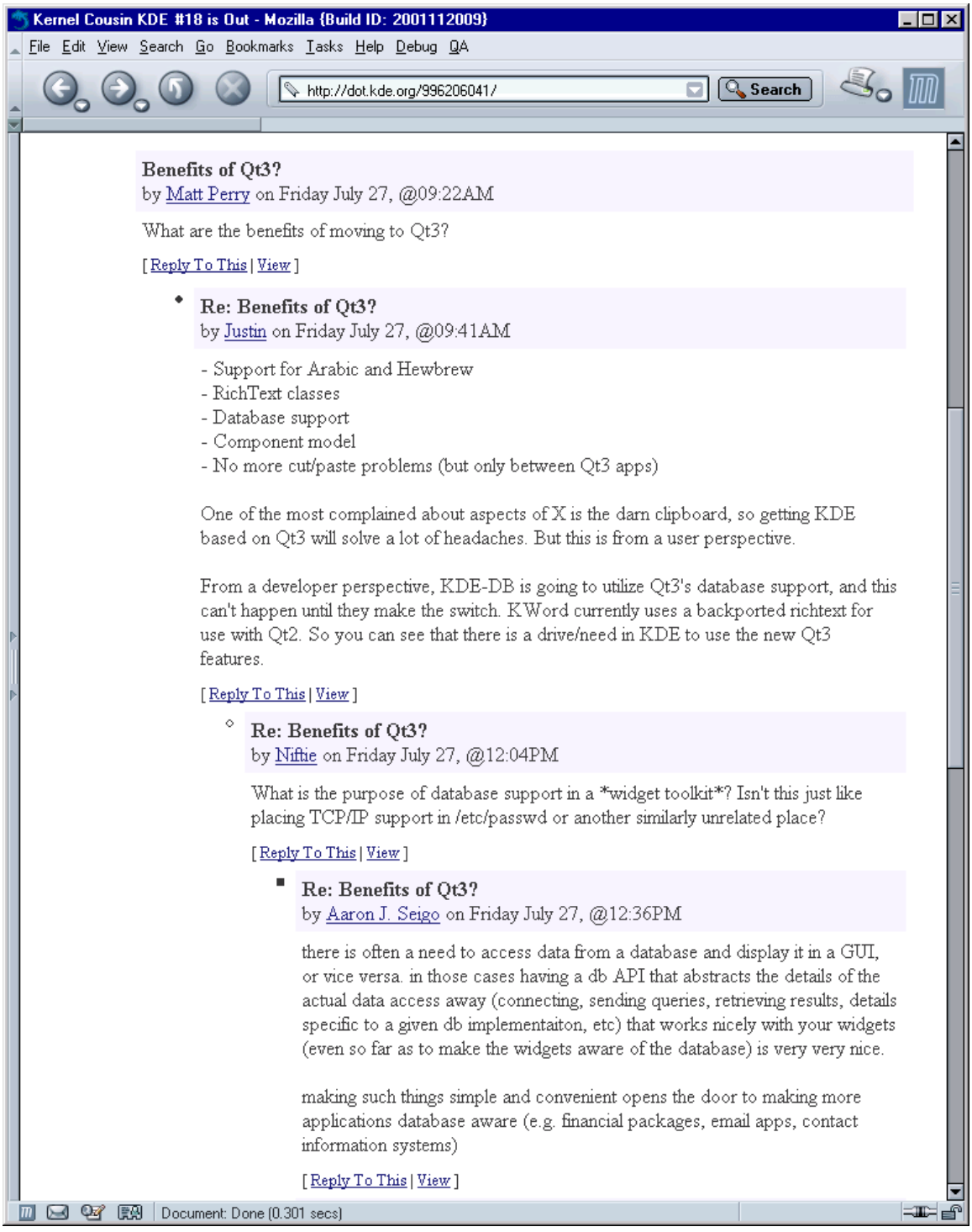


Exhibit 1. A sample of implicit requirements for the KDE software subsystem Qt3 expressed in a threaded email discussion. Source: http://dot.kde.org/996206041/, July 2001.

> Asserted system capabilities are *post hoc* requirements characterizing a functional capability that has already been implemented. The concerned developers justify their requirements through their provision of the required coding effort to make these capabilities operational. Senior members or core developers in the community then voted or agreed through discussion to include the asserted capability into the system's distribution [Fielding 1999]. The historical record 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 open source software requirement. This case displayed in Exhibit 2, finds a requirements "vision" 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 say through a process which relies on the use of system models represented as UML diagrams.

Perhaps community software development, and by extension, community development, are recognized as being important to the development and success of this system. It may also be a method for improving system quality and reliability when compared to existing software engineering tools and techniques (i.e., those based on UML, or supporting UML-based software design).

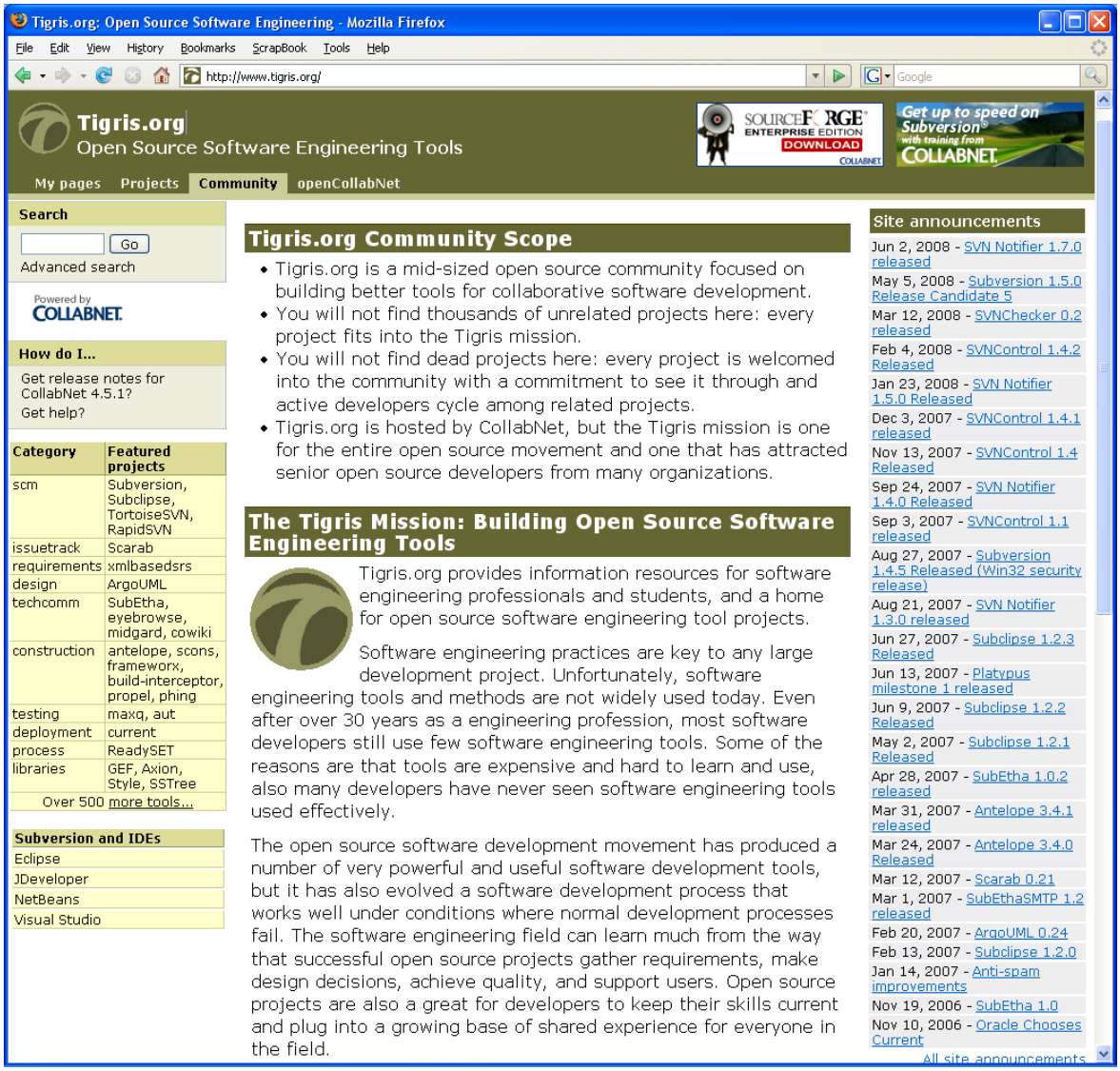


Exhibit 2. A software requirements vision statement encouraging both the development of software for the community and development of the community. Source: http://www.tigris.org, June 2008.

A third example reveals yet another kind of elicitation found in the Internet/Web infrastructure community. In Exhibit 3, 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. 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 2. The systems in Exhibits 2 and 3 must also be considered early in their overall development or maturity, since they call for functional capabilities that are needed to help make sufficiently complete for usage.

Thus, in understanding how the requirements of open source software 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 open source software engineering vision 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 open source software 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 an open source software development 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.

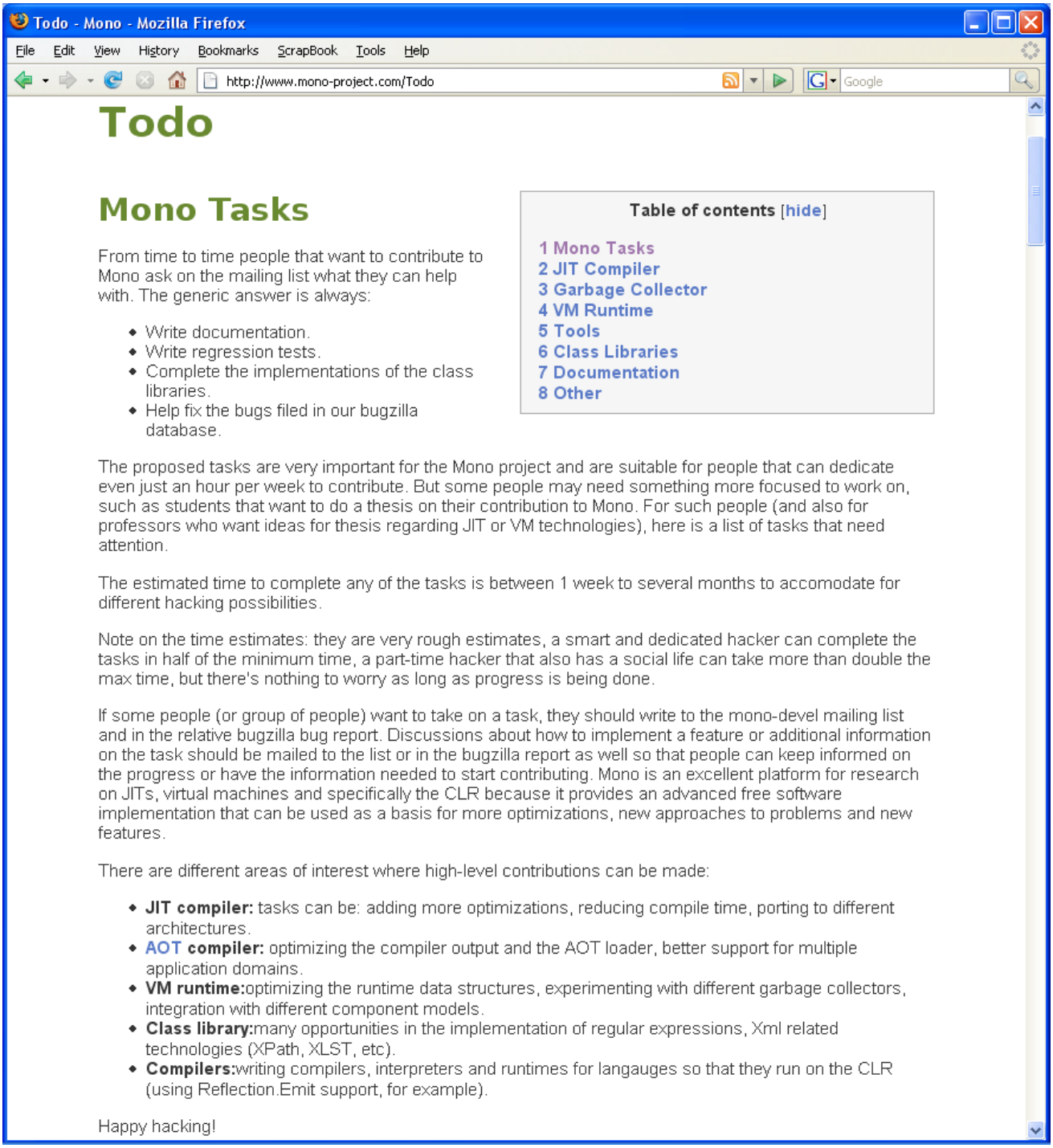


Exhibit 3: A non-functional requirement identifying a need for volunteers to become owners for yet to be developed software components. Source: http://www.mono-project.com/Todo, June 2008.

Exhibit 4. An asserted capability (in the center) that invites would-be open source software game developers to make new game mods, including improved versions, of whatever kind they require among the various types of available extensions. Source: http://www.ut3modding.com/, June 2008.

4.2 Requirements analysis vs. requirements reading, sense-making, and accountability

Software requirements analysis helps identify what problems a software system is suppose to address, while requirements specifications identify a mapping of user problems to system based solutions. In open source software development, 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 open source software requirements are analyzed.

Exhibits 5 and 6 come from different points in the same source document, a single research paper accessible on the Web, associated with the XXX-FOO research project. But how do software developers in this community (XXXXXXXX) understand what's involved in the functional operation of a complex system like this? One answer lies in the observation that developers who seek such an understanding must read this research paper quite closely, as well as being able to draw on their prior knowledge and experience in the relevant physical, telemetric, digital, and software domains [cf. Ackerman 2000]. A close reading likely means one that entails multiple re-readings and sense-making relative to one's expertise. A more casual though competent reading requires some degree of confidence and trust in the authors' account of how the functionality of the XXX architecture is configured, in order to accept what is presented as plausible, accurate, and correct.

The notion that requirements for open source software 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 XXX 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 1-4, 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 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 open source software requirements that has been described [Goguen 2000, Pavlicek 2000]. Analyzing open source software requirements entails little if any automated analysis, formal reasoning, or visual animation of software requirements specifications [cf. Nuseibeh 2000]. Yet, 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 open source software systems.

Exhibit 5. An asserted capability indicating that the requirements are very involved and complex.

Exhibit 6. **A specification of data-flow relationships among a network of software module pipelines that constitute the processing threads that must be configured.**

4.3 Requirements specification and modeling vs. continually emerging webs of software discourse

If the requirements for open source software systems are asserted rather than elicited, how are these requirements specified or modeled? In examining data from the four communities, of which Exhibits 1-6 are instances, it is becoming increasingly apparent that open source software requirements can emerge from the experiences of community participants through their 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 open source software system. This discourse is rendered in 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 Exhibit 5 from the X-ray and deep space imaging software community, we see passing reference in the opening paragraph to "the requirements for processing Chandra (remotely sensed) telemetry (imaging data) are very involved and complex." To comprehend and recognize what these involved and complex requirements are, community members who develop open source software for such applications will often be astrophysicists (with Ph.D. degrees), and rarely would be simply a competent software engineering professional. Subsequently, the astrophysicists that develop software in this community do not need to recapitulate any software system requirement that would be due to the problem domain (astrophysics). Instead, community members are already assumed to have mastery over such topics prior to software development, rather than encountering problems in their understanding of astrophysicists arising from technical problems in developing, operation, or functional enhancement of remote sensing or digital imaging software.

Thus, spanning the four communities and the six exhibits, we begin to observe that the requirements for open source software are specified in webs of discourse that reference or link:

- email or bboard discussion threads,

- system vision statements,
- ideas about system functionality and the non-functional need for volunteer developers to implement the functionality,
- promotional encouragement to specify and develop whatever functionality you need, which might also help you get a new job, and
- scholarly scientific research publications that underscore how the requirements of astronomical imaging 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 open source software requirements from new contributions, new contributors or participants, new ideas, new career opportunities, and new research publications.

4.4 Requirements validation vs. condensing discourse that hardens and concentrates system functionality and community development

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. Since open source software requirements are generally not recorded in a formal SRS document, nor are these requirements typically cast in a mathematical logic, algebraic, or state transition-based notational scheme, then how are the software implementations to be validated against their requirements?

In each of the four communities, it appears that the requirements for open source software are co-mingled 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 electronic documents including Web pages, sites, hypertext links, source code directories, threaded email transcripts, and more. In each community, requirements are 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. Goguen [Goguen 2000] 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 open source software. We find that requirements validation is a by-product, rather than an explicit goal, of how open source software requirements are constituted, described, discussed, cross-referenced, and hyperlinked to other informal descriptions of system and its implementations.

4.5 Communicating requirements vs. global access to open source software webs

One distinguishing feature of open source software associated with each of the four 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 open source software descriptions appear as conditions that do not receive much attention within the classic requirements engineering approaches, with few exceptions [Cybulski 1998]. Yet, each of these conditions helps in the

communication of open source software 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 1) or challenge the viability of emerging software requirements [cf. Conklin 1988, Lee 1990].

Each of the four communities also communicates community-oriented requirements. These non-functional requirements may seem similar to those for enterprise modeling [Nuseibeh 2000]. 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. Noll 1999]. Second, each community is interested in sustaining and growing the community's open source software 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 the ArgoUML community site, http://www.tigris.org.

4.6 Identifying a common foundation for the development of open source software 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 the requirements for open source software are developed. This centers of the emergent creation, usage, and evolution of informal software descriptions as the vehicle for developing open source software requirements.

5. Understanding open source software 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 FOSS system, project, or community requirements. FOSSD requirements (and subsequent system designs) are different. Thus, given the apparent success of sustained exponential growth for certain FOSS systems, and for the world-wide deployment of FOSSD practices, it is save to say that the ongoing development of FOSS systems points to the continuous development, articulation, adaptation, and reinvention of their requirements [cf. Scacchi 2006].

Second, the traditional virtues of high-quality software system requirements, namely, their consistency, completeness, traceability, and internal correctness are not so valued in FOSSD projects. FOSSD 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, FOSS 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.

Third, FOSS 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, FOSS requirements tend to be distributed across space, time, people, and the artifacts that interlink them. FOSS 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 FOSSD 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 and a centralized repository—that is, *centralized requirements*. Once again, FOSS projects represent an alternative paradigm to that long advocated by software engineering and software requirements engineering community.

Last, given that FOSS 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). FOSS capabilities embody requirements that have been found retrospectively to be both implementable and sustainable across releases. *Software capabilities specification* is thus perhaps a new engineering practice and methodology that can be investigated, modeled, supported, and refined in leading towards eventual 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 open source software development 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 open source software 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 open source software projects, though advocates who have do not report on the processes described here [DiBona 1999, Pavlicek 2000, Raymond 2001]. 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 [Davis 1990, Jackson 1995, Kotonya 1998, Nuseibeh 2000]. In contrast, this study sought to develop a baseline characterization of the how the requirements process for open source software occurs and the artifacts (and other mechanisms). Given such a baseline of the "as-is" process for open source software 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 [Scacchi 2000]. Such a position provides a basis for further studies which seek to examine how to redesign open source software 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 open source software development 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 open source software 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 open source software 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 open source software 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 [Davis 1990, Jackson 1995]. From the study presented here, it appears that open source software 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 open source software requirements. "Low" measures of such virtues might potentially point to increased likelihood of a failure to develop a sustainable open source software system. Subsequently, improving the quality of such virtues for open source software 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, open source software development practices are giving rise to a new view of how complex software systems can be constructed, deployed, and evolved. open source software development does not adhere to the traditional engineering rationality found in the legacy of software engineering life cycle models or prescriptive standards. The development open source software system requirements is inherently and undeniably a complex web of socio-technical processes, development situations, and dynamically emerging development contexts [Atkinson 2000, Goguen 2000, Kling 1982, Truex 1999, Viller 2000]. In this way, the requirements for open source software systems continually emerge through a web of community narratives. These extended narratives embody discourse that is captured in persistent, globally accessible, open source software informalisms that serve as an organizational memory [Ackerman 2000], hypertextual issue-based information system [Conklin 1988, Lee 1990], and a networked community environment for information sharing, communication, and social interaction [Kim 2000, 30, , Truex 1999]. 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 [Goguen 2000, Hine 2000, 19, Kling 1982 Nuseibeh 2000, Viller 2000]. 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, the Acquisition Research Program and the Center for the Edge Research Program at the Naval Postgraduate School. No endorsement implied. Chris Jensen, Thomas Alspaugh, John Noll,

Margaret Elliott, and other st the Institute for Software Research are collaborators on the research project described in this paper.

7. References

- ACKERMAN, M.S. and HALVERSON, C.A.: 'Reexamining Organizational Memory', *Communications ACM*, **43**, (1), pp. 59-64, January 2000.
- ATKINSON, C.J.: 'Socio-Technical and Soft Approaches to Information Requirements Elicitation in the Post-Methodology Era', *Requirements Engineering*, **5**, pp. 67-73, 2000.
- Bollinger, T., (2003). *Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense*, The MITRE Corporation, 2 January. Available at http://www.terrybollinger.com/dodfoss/dodfoss_html/index.html
- CLEVELAND, C.: 'The Past, Present, and Future of PC Mod Development', *Game Developer*, pp. 46-49, February 2001.
- CONKLIN, J. and BEGEMAN, M.L.: 'gIBIS: A Hypertext Tool for Effective Policy Discussion', *ACM Transactions Office Information Systems*, **6**, (4), pp. 303-331, October 1988.
- Crowston, K., & Howison, J. (2006). Hierarchy and centralization in Free and Open Source Software team communications. *Knowledge, Technology & Policy*, 18(4), 65–85.
- Crowston, K., Howison, J., & Annabi, H. (2006). Information systems success in Free and Open Source Software development: Theory and measures. *Software Process--Improvement and Practice*, 11(2), 123–148.
- Crowston, K., Wei, K., Li, Q., Eseryel, U. Y., & Howison, J. (2007). Self-organization of teams in free/libre open source software development. *Information and Software Technology Journal*, 49, 564-575.
- CYBULSKI, J.L. and REED, K.: 'Computer-Assisted Analysis and Refinement of Informal Software Requirements Documents', *Proceedings Asia-Pacific Software Engineering Conference* (APSEC'98), Taipei, Taiwan, R.O.C., pp. 128-135, December 1998.
- DAVIS, A.M.: '*Software Requirements: Analysis and Specification'*, Prentice-Hall, 1990.
- DIBONA, C. OCKMAN, S. and STONE, M.: '*Open Sources: Voices from the Open Source Revolution'*, O'Reilly Press, Sebastopol, CA, 1999.

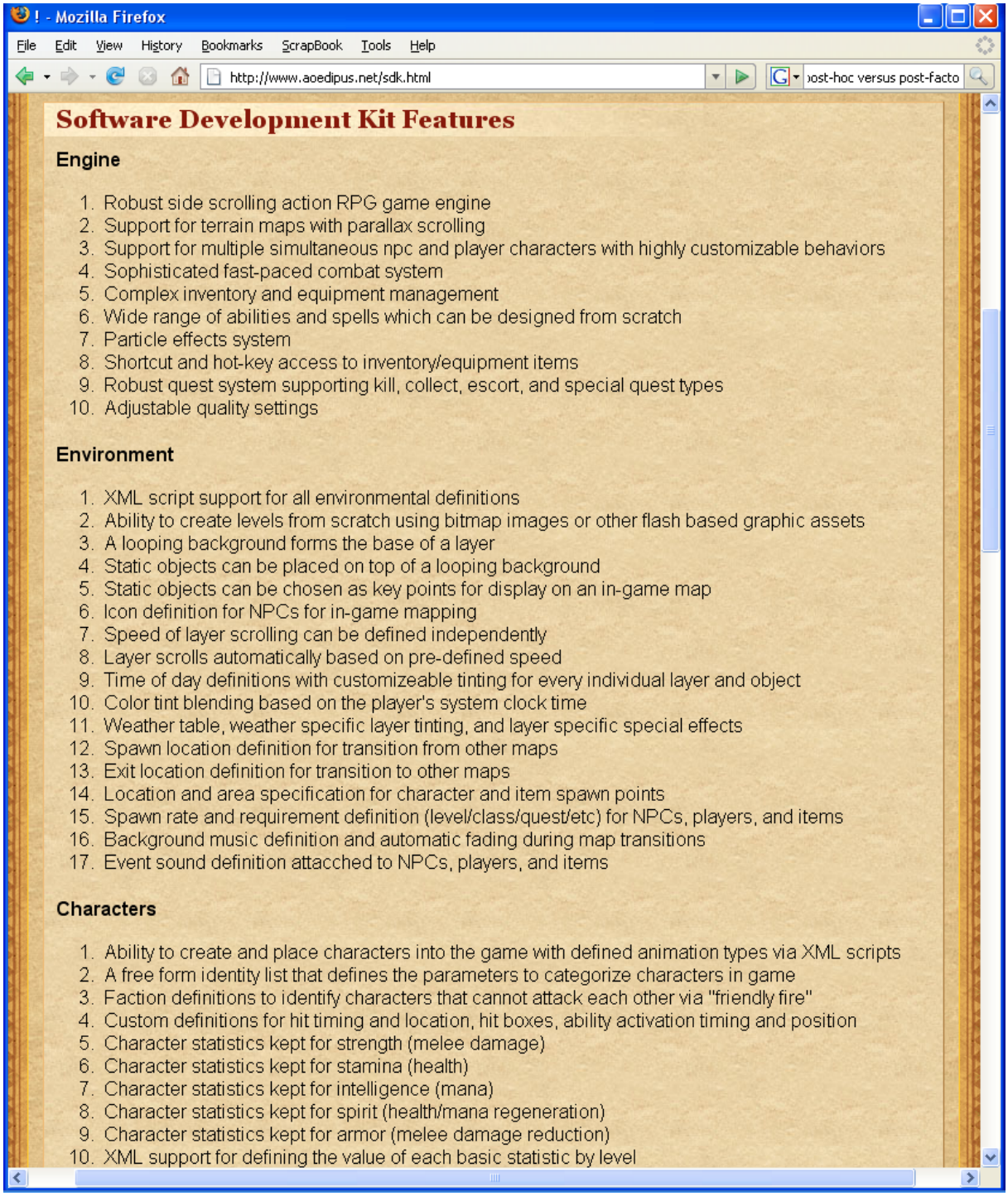
- Elliott, M. & Scacchi, W. (2005). Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture, in S. Koch (ed.), *Free/Open Source Software Development*, 152-172, IGI Publishing, Hershey, PA.
- Elliott, M. & Scacchi, W. (2008). Mobilization of Software Developers: The Free Software Movement, *Information, Technology and People*, 21(1), 4-33, 2008.
- Elliott, M., Ackerman, M.S. & Scacchi, W. (2007). Knowledge Work Artifacts: Kernel Cousins for Free/Open Source Software Development, *Proc. ACM Conf. Support Group Work (Group07),* Sanibel Island, FL, 177-186, November 2007.
- FIELDING,R.T.: 'Shared Leadership in the Apache Project', *Communications ACM,* **42**, (4), pp. 42-43, April 1999.
- Fogel, K., (1999). *Open Source Development with CVS*, Coriolis Press, Scottsdale, AZ.
- Fogel, K. (2005). *Producing Open Source Software: How to Run a Successful Free Software Project*, O'Reilly Press, Sebastopol, CA.
- Guertin N., (2007). (Director, Open Architecture, Program Executive Office IWS 7B). *Naval Open Architecture: Open Architecture and Open Source in DOD*, Presentation at "Open Source - Open Standards - Open Architecture," Association for Enterprise Integration Symposium, Arlington VA, 14 March 2007.
- GOGUEN, J.A.: 'Formality and Informality in Requirements Engineering (Keynote Address)', *Proc. 4th. Intern. Conf. Requirements Engineering,* pp. 102-108, IEEE Computer Society, 1996.
- HINE, C.: '*Virtual Ethnography'*, SAGE Publishers, London, 2000.
- Howison, J., Conklin, M., and Crowston, K. (2006). Flossmole: A collaborative repository for floss research, data, and analysis.. *Intern. J. Information Technology and Web Engineering*. 1(3):17-26.
- Howison, J., Wiggins, A. & Crowston, K. (2008). eResearch workflows for studying free and open source software development. In *Proc. 4th Intern. Conf. Open Source Software* (IFIP 2.13), Milan, Italy, 7-10 September.
- JACKSON, M.: '*Software Requirements & Specifications: Practice, Principles, and Prejudices'*, Addison-Wesley Pub. Co., Boston, MA, 1995.
- Jensen, C. & Scacchi,W. (2005). Process Modeling Across the Web Information Infrastructure, *Software Process--Improvement and Practice,* 10(3), 255-272, July-September 2005.

- Jensen, C. & Scacchi,W. (2007). Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study, in *Proc. 29th. Intern. Conf. Software Engineering,* Minneapolis, MN, ACM Press, May 2007, 364-374.
- Justice, Brig. General Nick (2007). (Program Executive Office, C3T), *Open Source Software Challenge: Delivering Warfighter Value*, Presentation at "Open Source - Open Standards - Open Architecture," Association for Enterprise Integration Symposium, Arlington VA, 14 March.
- Kwansik, B. and Crowston, K., (2005). Introduction to the special issue: Genres of digital documents*, Information, Technology and People,* 18(2).
- KIM, A.J.: '*Community-Building on the Web: Secret Strategies for Successful Online Communities'*, Peachpit Press, 2000.
- KLEIN, H. AND MYERS, M.D.: 'A Set of Principles for Conducting and Evaluating Intrepretive Field Studies in Information Systems', *MIS Quarterly*, **23**, (1), pp. 67-94, March 1999.
- KLING, R. and SCACCHI, W.: 'The Web of Computing: Computer technology as social organization'. In M. Yovits (ed.), *Advances in Computers*, **21**, pp. 3-90. Academic Press, New York, 1982.
- KOTONYA, G. and SOMMERVILLE, I.: '*Requirements Engineering: Processes and Techniques',* John Wiley and Sons, Inc, New York, 1998.
- Lanzara, G. F. and Morner, M. (2005). Artifacts rule! how organizing happens in open software projects. In Czarniawska, B. and Hernes, T., editors, *Actor Network Theory and Organizing*. Copenhagen Business School Press, Copenhagen.
- LEE, J.: 'SIBYL: a tool for managing group design rationale', *Proceedings of the Conference on Computer-Supported Cooperative Work*, Los Angeles, CA, ACM Press, pp. 79-92, 1990.
- Madey, G., Freeh, V., and Tynan, R., (2005). Modeling the F/OSS Community: A Quantitative Investigation, in S. Koch (ed.), *Free/Open Source Software Development*, 203-221, Idea Group Publishing, Hershey, PA.
- McDowell, P., Darken, R., Sullivan, J. and Johnson, E., (2006). 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. July.

- MI, P. and SCACCHI, W.: 'A Knowledge-based Environment for Modeling and Simulating Software Engineering Processes'. *IEEE Transactions on Knowledge and Data Engineering*, **2**, (3), pp. 283-294, Sept 1990.
- NOLL, J. and 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/
- NOLL, J. and W. SCACCHI.: 'Specifying Process-Oriented Hypertext for Organizational Computing', *J. Network and Computer Applications*, **24**, (1), pp. 39-61, 2001.
- NUSEIBEH, R. and EASTERBROOK, S.: 'Requirements Engineering: A Roadmap', in A. Finkelstein (ed.), *The Future of Software Engineering*, ACM and IEEE Computer Society Press, http://www.softwaresystems.org/future.html, 2000.
- PAVLICEK, R.: '*Embracing Insanity: Open Source Software Development'*, SAMS Publishing, Indianapolis, IN, 2000.
- RAYMOND, E.: '*The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary'*, O'Reilly and Associates, Sebastopol, CA, 2001.
- Riechers, C., (2007). (Principal Deputy, Asst. Sect. of the Air Force, Acquisition). *The Role of Open Technology in Improving USAF Software Acquisition*, Presentation at "Open Source - Open Standards - Open Architecture," Association for Enterprise Integration Symposium, Arlington VA, 14 March.
- Ripoche, G. and Gasser, L., (2003). Scalable Automatic Extraction of Process Models for Understanding F/OSS Bug Repair, *Proc. 16th Intern. Conf. Software Engineering & its Applications* (ICSSEA-03), Paris, France, December, 2003.
- Robinson, W., (2006). A Requirements Monitoring Framework for Enterprise Systems, *Requirements Engineering*, 11(1), 17-41.
- SCACCHI, W.: 'Understanding Software Process Redesign using Modeling, Analysis and Simulation , *Software Process--Improvement and Practice*, **5**, (2/3), pp. 183-195, 2000.
- Scacchi, W. (2002). Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 149(1), 24-39, February 2002.
- Scacchi, W. (2004). Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 21(1), 59-67, January/February 2004.

- Scacchi, W. (2005). Socio-Technical Interaction Networks in Free/Open Source Software Development Processes, in S.T. Acuña and N. Juristo (eds.), *Software Process Modeling*, 1-27, Springer Science+Business Media Inc., New York.
- Scacchi, W. (2006). Understanding Free/Open Source Software Evolution, in N.H. Madhavji, J.F. Ramil and D. Perry (eds.), *Software Evolution and Feedback: Theory and Practice*, 181-206, John Wiley and Sons Inc, New York, 2006.
- Scacchi, W. (2007). Free/Open Source Software Development: Recent Research Results and Methods, in M.V. Zelkowitz (ed.), *Advances in Computers*, 69, 243-295.
- Scacchi, W. (2007). Understanding the Development of Free E-Commerce/E-Business Software: A Resource-Based View, in S.K. Sowe, I. Stamelos, and I. Samoladas (eds.), *Emerging Free and Open Source Software Practices*, IGI Publishing, Hershey, PA, 170-190.
- Scacchi, W. (2008). Emerging Patterns of Intersection and Segmentation when Computerization Movements Interact in K.L. Kraemer and M. Elliott (eds.), *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, Information Today, Inc.
- Scacchi, W. & Alspaugh, T. (2008). Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense, *Proc. 5th Annual Acquisition Research Symposium*, Naval Postgraduate School, Monterey, CA.
- Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., & Lakhani, K. (2006). Understanding Free/Open Source Software Development Processes, *Software Process--Improvement and Practice*, 11(2), 95-105, March/April.
- Scacchi, W., Jensen, C., Noll, J. and Elliott, M. (2006). Multi-Modal Modeling, Analysis and Validation of Open Source Software Development Processes, *Intern. J. Internet Technology and Web Engineering,* 1(3), 49-63.
- SMITH, M. and KOLLOCK, P. (eds.): '*Communities in Cyberspace'*, Routledge, London, 1999.
- Spinuzzi, C., (2003). *Tracing Genres through Organizations: A Sociocultural Approach to Information Design*, MIT Press, Cambridge, MA.
- Starrett, E., (2007). Software Acquisition in the Army, *Crosstalk: The Journal of Defense Software Engineering*, 4-8, May.
- TRUEX, D., BASKERVILLE, R. and KLEIN, H.: 'Growing Systems in an Emergent Organization', *Communications ACM*, **42**, (8), pp. 117-123, 1999.

- VILLER, S. and SOMMERVILLE, I.: 'Ethnographically informed analysis for software engineers', *Int. J. Human-Computer Studies*, **53**, pp. 169-196, 2000.
- Weathersby, J.M., (2007). 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, June.
- Wheeler, B. (2007a). Open Source 2010: Reflections on 2007, *EDUCAUSE*, January/February 49-67.
- Wheeler, D.A., (2007b). Open Source Software (OSS) in U.S. Government Acquisitions, *The DoD Software Tech News*, 10(2), 7-13, June.
- YAMAGUCHI, Y., YOKOZAWA, M., SHINOHARA, T., and ISHIDA, T.: 'Collaboration with Lean Media: How Open-Source Software Succeeds', *Proceedings of the Conference on Computer Supported Cooperative Work,* (CSCW'00), pp. 329-338, Philadelphia, PA, ACM Press, December 2000.
- ZELKOWITZ, M.V. and WALLACE, D.: 'Experimental Models for Validating Technology', *Computer*, **31**, (5), pp. 23-31, May 1998.



Software Development Kit Features

Engine

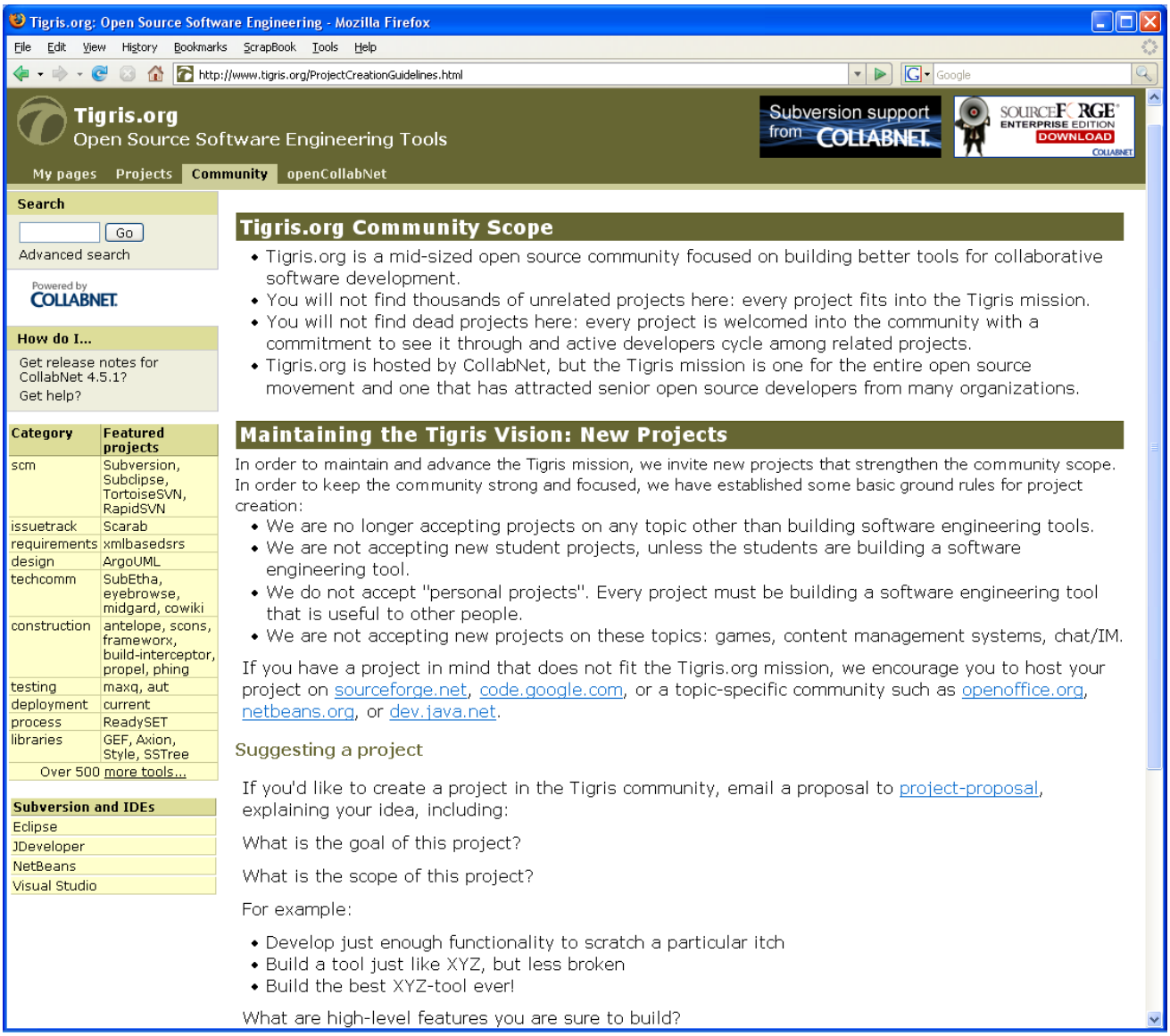
1. Robust side scrolling action RPG game engine
2. Support for terrain maps with parallax scrolling
3. Support for multiple simultaneous npc and player characters with highly customizable behaviors
4. Sophisticated fast-paced combat system
5. Complex inventory and equipment management
6. Wide range of abilities and spells which can be designed from scratch
7. Particle effects system
8. Shortcut and hot-key access to inventory/equipment items
9. Robust quest system supporting kill, collect, escort, and special quest types
10. Adjustable quality settings

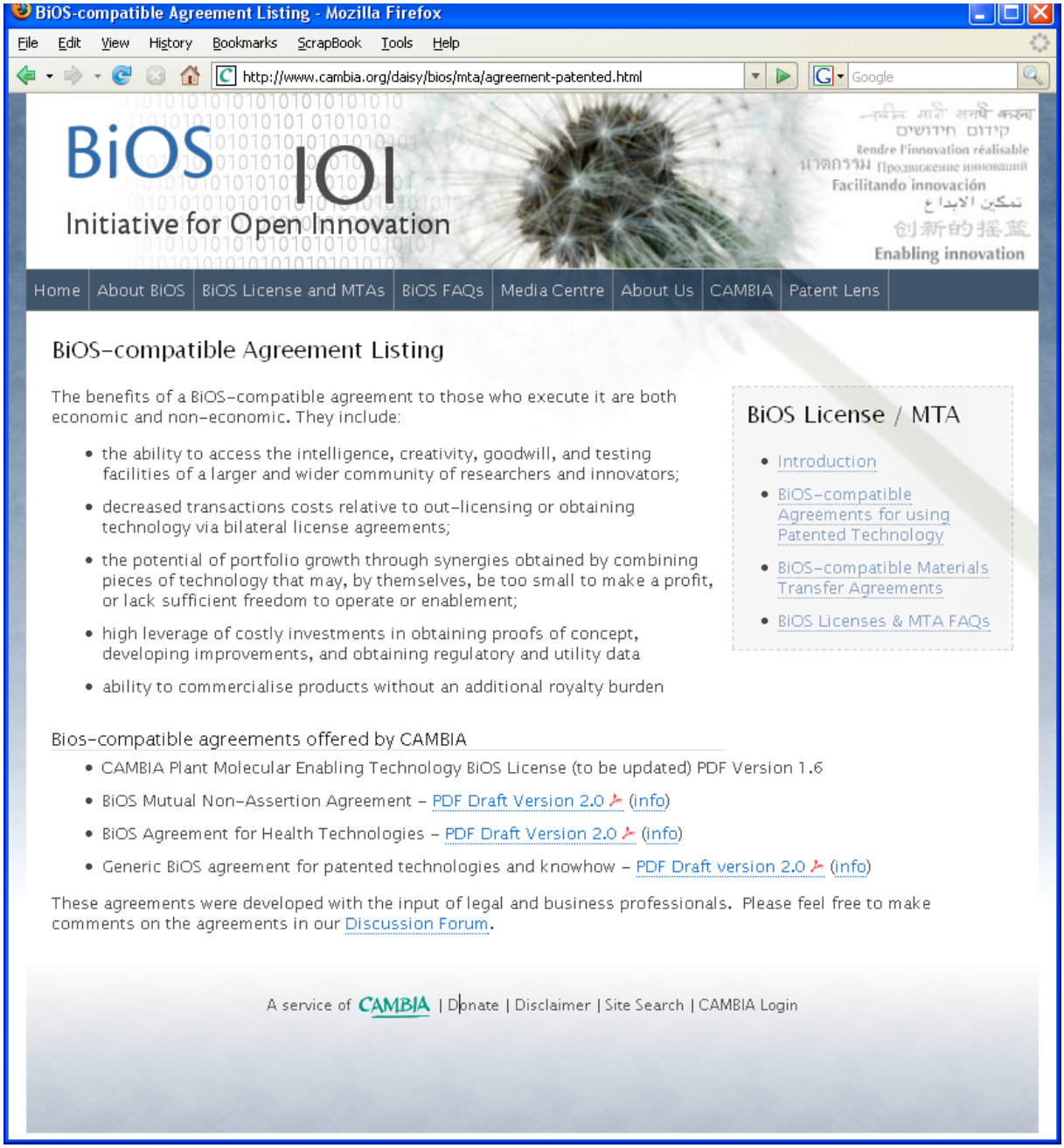
Environment

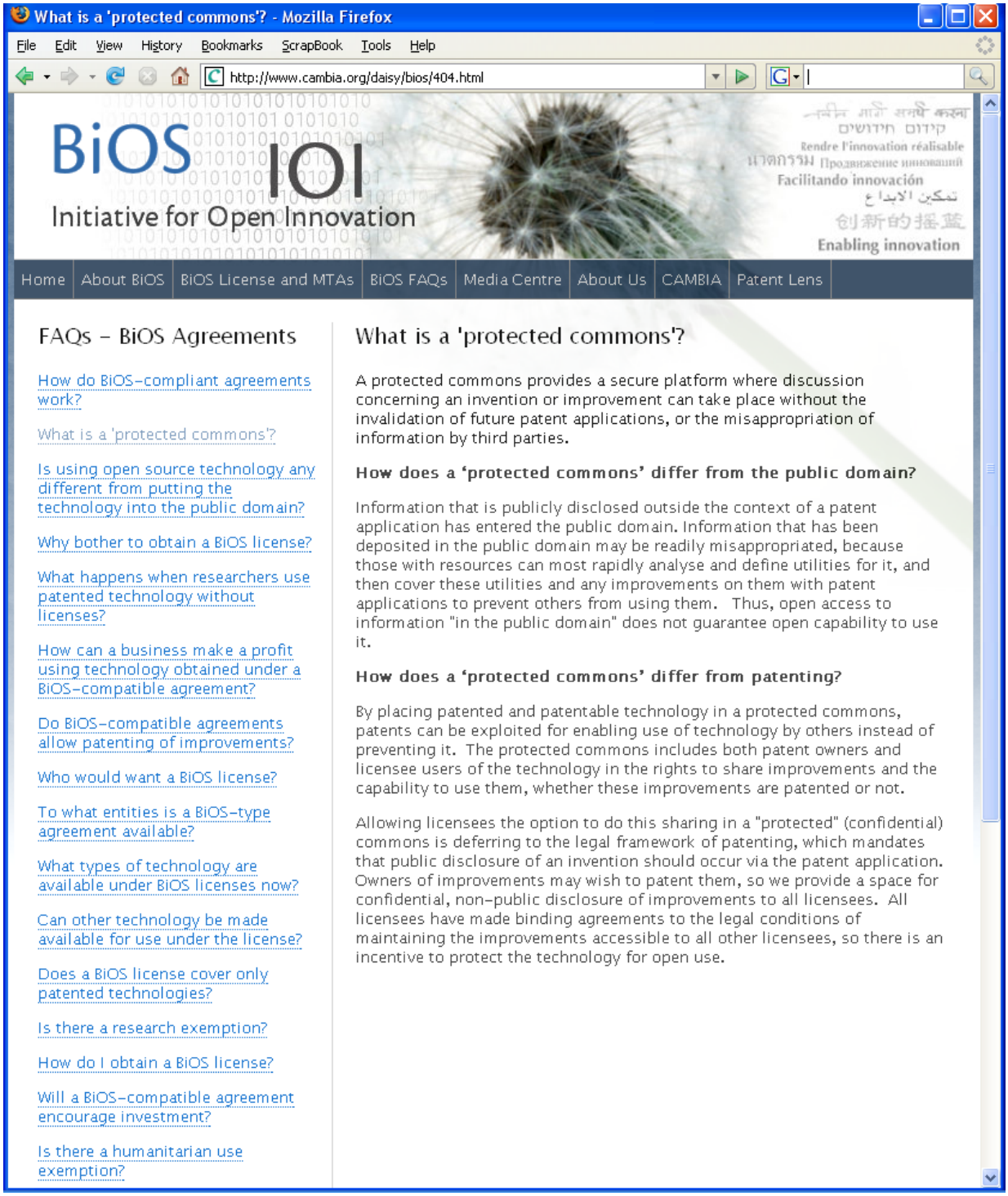
1. XML script support for all environmental definitions
2. Ability to create levels from scratch using bitmap images or other flash based graphic assets
3. A looping background forms the base of a layer
4. Static objects can be placed on top of a looping background
5. Static objects can be chosen as key points for display on an in-game map
6. Icon definition for NPCs for in-game mapping
7. Speed of layer scrolling can be defined independently
8. Layer scrolls automatically based on pre-defined speed
9. Time of day definitions with customizeable tinting for every individual layer and object
10. Color tint blending based on the player's system clock time
11. Weather table, weather specific layer tinting, and layer specific special effects
12. Spawn location definition for transition from other maps
13. Exit location definition for transition to other maps
14. Location and area specification for character and item spawn points
15. Spawn rate and requirement definition (level/class/quest/etc) for NPCs, players, and items
16. Background music definition and automatic fading during map transitions
17. Event sound definition attacched to NPCs, players, and items

Characters

1. Ability to create and place characters into the game with defined animation types via XML scripts
2. A free form identity list that defines the parameters to categorize characters in game
3. Faction definitions to identify characters that cannot attack each other via "friendly fire"
4. Custom definitions for hit timing and location, hit boxes, ability activation timing and position
5. Character statistics kept for strength (melee damage)
6. Character statistics kept for stamina (health)
7. Character statistics kept for intelligence (mana)
8. Character statistics kept for spirit (health/mana regeneration)
9. Character statistics kept for armor (melee damage reduction)
10. XML support for defining the value of each basic statistic by level


Tigris.org: Open Source Software Engineering - Mozilla Firefox
File Edit View History Bookmarks ScrapBook Tools Help
http://www.tigris.org/ProjectCreationGuidelines.html
Tigris.org
Open Source Software Engineering Tools
Subversion support from COLLABNET
SOURCEFORGE ENTERPRISE EDITION DOWNLOAD
My pages Projects Community openCollabNet
Search
Go
Advanced search
Powered by COLLABNET
How do I...
Get release notes for CollabNet 4.5.1?
Get help?
Category | Featured projects
scm | Subversion, Subclipse, TortoiseSVN, RapidSVN
issuetrack | Scarab
requirements | xmlbasedsrs
design | ArgoUML
techcomm | SubEtha, eyebrowse, midgard, cowiki
construction | antelope, scons, frameworx, build-interceptor, propel, phing
testing | maxq, aut
deployment | current
process | ReadySET
libraries | GEF, Axion, Style, SSTree
Over 500 more tools...
Subversion and IDEs
Eclipse
JDeveloper
NetBeans
Visual Studio
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.
Maintaining the Tigris Vision: New Projects
In order to maintain and advance the Tigris mission, we invite new projects that strengthen the community scope. In order to keep the community strong and focused, we have established some basic ground rules for project creation:
We are no longer accepting projects on any topic other than building software engineering tools.
We are not accepting new student projects, unless the students are building a software engineering tool.
We do not accept "personal projects". Every project must be building a software engineering tool that is useful to other people.
We are not accepting new projects on these topics: games, content management systems, chat/IM.
If you have a project in mind that does not fit the Tigris.org mission, we encourage you to host your project on sourceforge.net, code.google.com, or a topic-specific community such as openoffice.org, netbeans.org, or dev.java.net.
Suggesting a project
If you'd like to create a project in the Tigris community, email a proposal to project-proposal, explaining your idea, including:
What is the goal of this project?
What is the scope of this project?
For example:
Develop just enough functionality to scratch a particular itch
Build a tool just like XYZ, but less broken
Build the best XYZ-tool ever!
What are high-level features you are sure to build?


BiOS-compatible Agreement Listing - Mozilla Firefox
File Edit View History Bookmarks ScrapBook Tools Help
http://www.cambia.org/daisy/bios/mta/agreement-patented.html
Google
BiOS IOI
Initiative for Open Innovation
Enabling innovation
Home About BiOS BiOS License and MTAs BiOS FAQs Media Centre About Us CAMBIA Patent Lens
BiOS-compatible Agreement Listing
The benefits of a BiOS-compatible agreement to those who execute it are both economic and non-economic. They include:
the ability to access the intelligence, creativity, goodwill, and testing facilities of a larger and wider community of researchers and innovators;
decreased transactions costs relative to out-licensing or obtaining technology via bilateral license agreements;
the potential of portfolio growth through synergies obtained by combining pieces of technology that may, by themselves, be too small to make a profit, or lack sufficient freedom to operate or enablement;
high leverage of costly investments in obtaining proofs of concept, developing improvements, and obtaining regulatory and utility data
ability to commercialise products without an additional royalty burden
BiOS License / MTA
Introduction
BiOS-compatible Agreements for using Patented Technology
BiOS-compatible Materials Transfer Agreements
BiOS Licenses & MTA FAQs
Bios-compatible agreements offered by CAMBIA
CAMBIA Plant Molecular Enabling Technology BiOS License (to be updated) PDF Version 1.6
BiOS Mutual Non-Assertion Agreement – PDF Draft Version 2.0 (info)
BiOS Agreement for Health Technologies – PDF Draft Version 2.0 (info)
Generic BiOS agreement for patented technologies and knowhow – PDF Draft version 2.0 (info)
These agreements were developed with the input of legal and business professionals. Please feel free to make comments on the agreements in our Discussion Forum.
A service of CAMBIA | Donate | Disclaimer | Site Search | CAMBIA Login


What is a 'protected commons'? - Mozilla Firefox
File Edit View History Bookmarks ScrapBook Tools Help
http://www.cambia.org/daisy/bios/404.html
BiOS
Initiative for Open Innovation
Rendre l'innovation réalisable
Facilitando innovación
Enabling innovation
Home
About BiOS
BiOS License and MTAs
BiOS FAQs
Media Centre
About Us
CAMBIA
Patent Lens
FAQs – BiOS Agreements
How do BiOS-compliant agreements work?
What is a 'protected commons'?
Is using open source technology any different from putting the technology into the public domain?
Why bother to obtain a BiOS license?
What happens when researchers use patented technology without licenses?
How can a business make a profit using technology obtained under a BiOS-compatible agreement?
Do BiOS-compatible agreements allow patenting of improvements?
Who would want a BiOS license?
To what entities is a BiOS-type agreement available?
What types of technology are available under BiOS licenses now?
Can other technology be made available for use under the license?
Does a BiOS license cover only patented technologies?
Is there a research exemption?
How do I obtain a BiOS license?
Will a BiOS-compatible agreement encourage investment?
Is there a humanitarian use exemption?
What is a 'protected commons'?
A protected commons provides a secure platform where discussion concerning an invention or improvement can take place without the invalidation of future patent applications, or the misappropriation of information by third parties.
How does a 'protected commons' differ from the public domain?
Information that is publicly disclosed outside the context of a patent application has entered the public domain. Information that has been deposited in the public domain may be readily misappropriated, because those with resources can most rapidly analyse and define utilities for it, and then cover these utilities and any improvements on them with patent applications to prevent others from using them. Thus, open access to information "in the public domain" does not guarantee open capability to use it.
How does a 'protected commons' differ from patenting?
By placing patented and patentable technology in a protected commons, patents can be exploited for enabling use of technology by others instead of preventing it. The protected commons includes both patent owners and licensee users of the technology in the rights to share improvements and the capability to use them, whether these improvements are patented or not.
Allowing licensees the option to do this sharing in a "protected" (confidential) commons is deferring to the legal framework of patenting, which mandates that public disclosure of an invention should occur via the patent application. Owners of improvements may wish to patent them, so we provide a space for confidential, non-public disclosure of improvements to all licensees. All licensees have made binding agreements to the legal conditions of maintaining the improvements accessible to all other licensees, so there is an incentive to protect the technology for open use.