

Understanding Software Requirements: Lessons from Free/Open Source Software Development Projects

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA
Wscacchi @ ics <dot> uci <dot> edu
<http://www.ics.uci.edu/~wscacchi>
March 2007

Overview

Worldwide, free/open source software development (FOSSD) projects now number in the tens of thousands, while hundreds have gone on to experience sustained exponential growth [Scacchi 2006]. What does this tell us about what's going on in FOSSD projects? What kinds of requirements development process can give rise to or accommodate sustained exponential growth of software systems? The sustained growth of FOSS systems appears to defy the laws of software evolution that have long predicted software system growth following an inverse-square growth curve, due in part to the stabilization of their requirements during development, followed by a long period of maintenance and incremental requirements adaptation.

Software requirements analysis helps identify what problems a software system is suppose to address, while requirements specification identify an initial mapping of problems to system based solutions. In FOSSD projects, how does requirements analysis occur, and where and how are requirements specifications described? Studies to date have yet to report discovery of records for formal requirements elicitation, capture, analysis, and validation activity of the kind suggested by modern software engineering textbooks [cf. Sommerville 2004]. In general, they cannot be found online on F/OSS Web sites, or offline in published technical reports or documents identified as "requirements specification" documents, with few exceptions. What has been found and observed through ethnographic study [cf. Viller and Sommerville 2000] of FOSSD projects in computing worlds for networked games [Scacchi 2004], astrophysics, Internet infrastructure applications and others [Scacchi 2002] is different, as described here.

In current practice, FOSS requirements are manifested as threaded messages or discussions that are captured and/or posted on a Web site for open review, elaboration, refutation, or refinement. Requirements analysis and specification are *implied activities* that routinely emerge as a by-product of community discourse about what their software should or should not do, as well as who will take responsibility for contributing new or modified system functionality that effectively creates a system's requirements. FOSS requirements appear in the form of after-the-fact assertions within private and public

email discussion threads, ad hoc software artifacts (e.g., source code fragments included within a message) and Web site content updates that continually emerge [Scacchi 2002, Truex 1999].

Furthermore, developing FOSS requirements is a *community building process* that must be institutionalized both within a community and its software informalisms to flourish. In this regard, the development of FOSS requirements is not a traditional requirements engineering process, at least, not yet. It is instead socio-technical process that entails the development of constructive social relationships, informally negotiated social agreements, and a commitment to participate through sustained contribution of software discourse and shared representations, much like the other processes identified above. Thus, community building and sustaining participation are essential and recurring activities that enable FOSS to persist without central corporate authority.

Findings from FOSSD studies

Software informalisms are the information resources and artifacts that participants use to describe, proscribe, or prescribe what's happening in a FOSSD project. 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.

The most common informalisms used in FOSSD projects include (i) communications and messages within project Email, (ii) threaded message discussion forums, bulletin boards, or group blogs, (iii) news postings, (iv) project digests, and (v) instant messaging or Internet relay chat. They 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. FOSS (xiii) project property licenses (e.g., the GPL v.3) 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 plug-in externally developed software modules, 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 identified above 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., SourceForge.net, Savannah.org, Freshment.org, Tigris.org, Apache.org, Mozilla.org), community software Web sites (e.g., 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 (e.g., CVS or Subversion), and (xxii) software bug reports and (xxiii) issue tracking data base like Bugzilla (see www.bugzilla.org/).

Together, these two dozen or so 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.

Multi-project FOSS Web sites also serve as hubs or “community cores” that centralize attention for what is happening with the development of focal FOSS systems, their status, participants and contributors, discourse on pending/future needs, etc. Furthermore, by their very nature, these Web sites are generally global in reach and publicly accessible. This means the potential exists for contributors to come from multiple remote sites (geographic dispersion) at different times (24/7), from multiple nations, representing the interests of multiple cultures or ethnicity. Thus, multi-project FOSS Web sites help to make visible online virtual organizations, inter-project alliances, community and social networks that can share resources, artifacts, interests, and source code.

All of these conditions point to *new kinds of requirements for FOSSD projects*—for example, community building requirements, community software requirements, and community information sharing system (Web site and interlinked communication channels for email, forums, and chat) requirements [Scacchi 2002, Truex, 1999]. These requirements may entail both functional and non-functional requirements, but they will most typically be expressed using FOSS informalisms, rather than using formal notations based on some system of mathematical logic known by few.

More conventionally, requirements analysis, specification, and validation are not practiced nor valued as a necessary task in F/OSS projects that produces a mandatory requirements deliverable. But what can be observed are widespread practices that reveal *FOSS requirements development entails reading and sense-making of online content, and also interlinked webs of discourse that effectively trace, condense and harden into retrospective software requirements*. This arises all while being globally accessible to existing, new or former F/OSS project participants. FOSS requirements thus appear in most clearly in hindsight, yet are continuously emerging and (re)negotiated in foresight.

Summary observations about software requirements

Building from the findings about the development of FOSS system requirements, the following observations and interpretations can be offered.

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 safe to say that the ongoing development of FOSS systems points to the continuous development, articulation, adaptation, and reinvention of their requirements [cf. Gasser 2003].

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 provisions” (and not software provisioning, which refers to software release and distribution activities). FOSS provisions embody requirements that have been found retrospectively to be both implementable and sustainable across releases. *Software provisions engineering* is thus

perhaps a new engineering practice and methodology that can be investigated, modeled, supported, and refined in leading towards eventual principles for software system provisions.

References

L. Gasser, W. Scacchi, G. Ripoché, and B. Penne, Understanding Continuous Design in F/OSS Projects, *Proc. 16th Intern. Conf. Software & Systems Engineering and their Applications*, Paris, December 2003.

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

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

W. Scacchi, Understanding Free/Open Source Software Evolution, in N.H. Madhavji, J.F. Ramil and D. Perry (eds.), *Software Evolution and Feedback: Theory and Practice*, John Wiley and Sons Inc, New York, 181-206, 2006.

W. Scacchi, Free/Open Source Software Development: Recent Research Results and Methods, in M.V. Zelkowitz (ed.), *Advances in Computers*, 69, 243-269, 2007.

I. Sommerville, *Software Engineering* 7th Edition, Addison-Wesley, New York 2004.

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

S. Viller and I. Sommerville, Ethnographically Informed Analysis for Software Engineers, *Intern. J. Human-Computer Studies*, 53, 169-196, 2000.