Understanding Continuous Design in F/OSS Projects

Les Gasser[1,2]
gasser@uiuc.edu

Walt Scacchi[2]
wscacchi@ics.uci.edu

Gabriel Ripoche[1, 3]
gripoche@uiuc.edu

Bryan Penne[1]
bpenne@uiuc.edu

[1] Graduate School of Library and Information Science
University of Illinois at Urbana-Champaign
501 E. Daniel St., Champaign, IL 61820, USA
Phone: +1 217 265 5021 / Fax: +1 217 244 3302

[2] Institute for Software Research
University of California at Irvine
ICS2 242, UCI, Irvine, CA, 92697-3425
Phone: +1 949 824 4130 / Fax: +1 949 824 1715

[3] LIMSI-CNRS / Université Paris XI
BP 133, 91403 Orsay Cedex, France
Phone: +33 1 69 85 81 01 / Fax: +33 1 69 85 80 88

Abstract

Open Source Software (OSS) is in regular widespread use supporting critical applications and infrastructure, including the Internet and World Wide Web themselves. The communities of OSS users and developers are often interwoven. The deep engagement of users and developers, coupled with the openness of systems lead to community-based system design and re-design activities that are continuous. Continuous redesign is facilitated by communication and knowledge-sharing infrastructures such as persistent chat rooms, newsgroups, issue-reporting/tracking repositories, sharable design representations and many kinds of "software informalisms." These tools are arenas for managing the extensive, varied, multimedia community knowledge that forms the foundation and the substance of system requirements. Active community-based design processes and knowledge repositories create new ways of learning about, representing, and defining systems that challenge current models of representation and design. This paper presents several aspects of our research into continuous, open, community-based design practices. We discuss several new insights into how communities represent knowledge and capture requirements that derive from our qualitative empirical studies of large (ca. 2GB+) repositories of problem-report data, primarily from the Mozilla project.

Keywords

Continuous design; Open source software; Knowledge management; Knowledge representation; Community knowledge; Specification.

1 Introduction

In current research we are studying software maintenance work, bug reporting, and repair in Free/Open Source Software (F/OSS) communities, giving special attention to factors such as knowledge exchange, effectiveness of tool support, social network structures, and organization of project activity. More specifically, we are examining how these factors affect such outcome variables as the time taken to resolve reported problems, the ability to detect interdependent and duplicate problems, the effective scope of repairs (the degree to which the community can address the entire range of software problems that appear, rather than just selected subsets), and the general quality of software.

Answering these questions is important to a number of scientific, government, or industrial communities. U.S. science agencies continue to make substantial (multi-million dollar per year) investments in complex software systems supporting research in natural and physical sciences (e.g., Bioinformatics, National Virtual Observatory) via computational science test-beds (e.g., Tera-Grid, CyberInfrastructure), which software is intended as free/open source [13,10]. E-Government initiatives around the world are increasingly advocating or mandating the use of F/OSS in their system design and development efforts [5,8]. Also, industrial firms in the U.S. that develop complex software systems for internal applications or external products are under economic pressure to improve their software productivity and quality, while reducing their costs. F/OSS is increasingly cited as one of the most promising and most widely demonstrated approach to the reuse of software [4] in ways that can make the design and development of complex software faster, better, and cheaper [22].

However, in each of these cases, there is no existing framework or guideline for how to design, manage, or evaluate F/OSS software systems and processes. Moreover we don't have a clear picture of how these "consumers" might best expend their scarce resources for continuous system design and redesign [16], or how they might redesign their own processes to assess and take advantage of F/OSS technologies [15].

Our research process is based on qualitative and computational analysis of large corpuses of longitudinal data from Web sites, public repositories, online discussion forums, and online artifacts from F/OSS development projects. Projects analyzed include networked computer games, Internet/Web infrastructure, X-ray astronomy/deep space imaging, and academic software research. We have access to a number of large collections of current data from these projects, comprising at least six repositories of problem report/analysis activity, with a total of over 500,000 reports, containing approximately 5 million individual comments, and involving over 60,000 reporters and developers.

We are using both human-based grounded-theory and computational methods (such as automated concept extraction, data and text-mining, process modeling) to empirically discover, identify and comparatively analyze patterns of continuous software design in the projects under study [14]. More specifically, we are looking for design episodes, basic design processes (such as collective sense making, negotiation, information seeking, and conflict management), design-management activities, and design-support/design-management infrastructure.

For example, we are using statistical text-analysis tools to automatically extract episodes of design activity when traces of those activities can be characterized with specific "language models". We are also developing ways of automatically extracting and coding change data from large corpuses to mechanically develop explicit, generalized process models of design processes, and link specific steps or paths in these process models back to the underlying data from which they are derived.

2 Continuous Design in F/OSS

In doing this research we have begun to find that the work represented in these repositories goes far beyond software repair and maintenance ("bug fixing"). Concepts such as "bug" and "repair" connote some deviation from an accepted standard of function or behavior, and restoration of performance to that standard. However, it seems that much of the work we see in our data involves *continuous distributed collective software specification and design* [7] instead of bug fixing. We are seeing some clear trends that run somewhat contrary to conventional wisdom and normative software development practice for large systems, for example:

1. Specifications of software function, usability, structure, etc. are not (and in fact cannot be) known when software is designed and released. Instead, software artifacts capture and track the emerging preferences of their co-emerging user communities in a continuing cycle of innovation. Explicit, formal specifications or formal design processes almost never exist. The code itself, coupled with persistent traces of extended discussions and debates, constitute the current and evolving "specification".
2. Software builders rely on shared collections of "software development informalisms" [17]—which are assemblages of semi-structured and relatively informal knowledge-sharing tools, representations and practices—instead of rather more formal development processes and specifications.

We believe this is a fundamentally new type of software design process [2], as it effectively links many interacting, continuous, open, collective processes—which could easily become chaotic—to produce the surprisingly efficient, stable, useful, circumscribed artifacts that result from many F/OSS projects. Our data on these patterns of design also suggests that specific software system designs and design management practices co-evolve with the emerging teamwork patterns, broader community-wide design practices, and the collective discourse that constitutes the meaning and embodies the purpose for how the F/OSS systems are built. In fact, what we are seeing here is a type of "emergent knowledge process" [9].

These observations are significant because this is not a prediction that follows from the current principles of software design, software engineering, and software evolution [21], which generally see a sequential (waterfall model), or an iterative and incremental (spiral model) design process and development life cycle [18]. In contrast, we observe that F/OSS software is produced in a manner that can be more productive, higher quality, and lower cost than current principles of software engineering suggest [22].

These observations are also surprising, because they suggest that even though F/OSS development projects don't rely on explicit representations or formal notations for designs [24], they can achieve comparable or better results by relying on specific patterns of social interaction and discourse to mature and harden informal or conversational system descriptions into stable, sustainable, widely used software systems of acceptable quality to their user-developers [6,23,17,19]. Thus these F/OSS projects have no need to "fake" a rational design process, as is all too common in software engineering [12]. Instead, F/OSS projects seem to rely on enacting a collective capability for distributed design and organizing [11].

The question that emerges from these previous observations is the following: how does a globally dispersed community of F/OSS developers design and redesign complex software systems in a continuously emergent manner? Early analysis of these continuous design processes has led us to two hypotheses about how communities understand and represent knowledge of the artifacts they "design in use" [3].

3 Continuous Design and Community Knowledge

3.1 Construction of Community Knowledge

Community knowledge of software artifacts is transformed from linear/learned to ***continuous/constructed.***

A standard software process of "specify, design, build, test, release, maintain" effectively separates users at the point of release from the specification/design process and means that they can only indirectly learn about artifacts: knowledge of design, function, and structures is communicated to users through manuals, training, instructions, etc. In the ongoing design processes we are seeing, the community *continuously constructs* its artifact knowledge in concert with the development of the artifact itself. Using initial seeds and prototypes, the community continuously, collectively develops and transforms its knowledge of what the target artifact is, how it works, what it should be, and how it should work. What is not yet well understood are the specific processes of this development and transformation. However, we observed the following points:

Community knowledge is constructed from a multiplicity of viewpoints, representations, and experiences

Because of the wide variety of people participating in F/OSS projects (users, "power" users, developers, testers, etc.), the knowledge development/transformation process is characterized by *extreme multiplicity* of viewpoints, representations, experiences, and usages.

Community knowledge development is enacted through knowledge management environments

F/OSS design/development processes are made possible and enacted by shared, structured, and (more or less) complex knowledge management environments that persistently capture traces of design and analysis processes. These systems—which range from simple newsgroups to complex problem repositories and compiled documents—allow the community to accumulate, organize, make sense of, and use the wide variety of knowledge contributed by the many people involved.

The artifact (and knowledge about it) is constructed through a dynamic network of social processes

The relatively "linear with feedback loops" structure of the more standard software development process is replaced by a network of (largely social) processes arranged in a highly dynamic topology, including: design, construction, testing, releasing, work coordination, critiquing, use, suggesting, specification, tool-building, triage, negotiation, evaluation, etc. Compared to processes represented in a rational design scenario, processes constituting F/OSS design/development are varied and numerous, and dynamically and loosely articulated.

3.2 Representation of Community Knowledge

Community representation of knowledge is transformed from bounded, faithful reflections to ***massive, contentious assemblages****.*

Standard models of knowledge representation underlying problem report systems assume a conventional relation between problem reports on the one hand, and the use and testing experiences they represent on the other hand. In this view, *documents* bear a *relationship* to an experiential *life-world*, and documents represent things, events, and experiences through this relationship. Empirical examination of the actual processes of continuous collective software design is instead revealing a quite different underlying representation model.

Documents record viewpoints, and these viewpoints are subject to multiple interpretations

Many comments are typically made on each problem report (an average of 10 comments per report in Bugzilla[1]). These comments often contradict or conflict with each other, and some comments are not interpretable by some readers and respondents. Moreover problems are reported more than once in different ways, and it is often difficult to correlate or link these reports and their underlying manifestations.

The scope of documentation is unknowable

Duplicate bug reports are not always linked, index terms differ, and since the documentation system itself is a large, open system, its state at any moment is being revised. Therefore, knowledge is incomplete in the sense that there might be "more out there", but that this potential knowledge is not accessible to the community as a whole. For example, separate groups might work for months on related problems (even identical problems sometimes) before realizing that their work is related and creating the "missing link".

The fidelity of relationship between documents and experiences is uncertain

Problem reports are textual discourses that purport to describe aspects of problems; they are not the problems themselves. Numerous debates in the Bugzilla corpus involve negotiation over the "proper" interpretation or referent of a problem report (e.g. Which underlying cause or manifestation does it refer to? Which experience is "real" and which is incorrect?) Thus it is in general not a routine matter to directly and unambiguously match a problem description with an experience or a cause.

The relevant life-world is unknowable

Users have widely varying styles of use and functional requirements, which are in general impossible to fully describe in a distributed context. This multiplicity, diversity, and distribution together mean that the scope of experience underlying problem descriptions is not knowable.

Taken together, these four observations mean that standard notions of representation fail to capture what is really going on in collective continuous design contexts such as the F/OSS projects we are studying, and new models are needed.

4 Conclusions

These results provide new perspectives on community-wide practices of capturing and managing software design knowledge, on how knowledge is articulated, breaks down and is reconstituted, and on how it is shared and propagated. This research provides a conceptual shift in understanding how system development and use are bound together with the richness, variety, and temporal evolution of the socio-technical contexts provided by the global software industry.

Acknowledgements

We gratefully acknowledge the contributions made by Bob Sandusky. This material is based upon work supported by the National Science Foundation under Grant No. 0205346. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

[1] Statistic obtained from a snapshot of Bugzilla taken in March 2002 (over 128,000 bug reports).

References

[1] Augustin, L., Bressler, D., Smith, G., Accelerating Software Development through Collaboration. In Proceedings of the 24th International Conference on Software Engineering, Orlando, FL, pp 559-563. May 2002.

[2] Benkler, Y., Coase's Penguin, or Linux and the Nature of the Firm. 112 Yale Law Journal (2002-03). http://www.benkler.org/CoasesPenguin.html.

[3] Bodker, S., Computer Applications as Mediators of Design and Use - A Developmental Perspective. Report DAIMI PB-542, Computer Science Department, Aarhus University. October 1999.

[4] Brown, A.W., Booch G., Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors. In Proceedings of the 7th International Conference on Software Reuse. Lecture Notes in Computer Science, vol. 2319, pp 123-136. 2002.

[5] EGovOS, The Center for Open Source & Government. http://www.egovos.org. 2003.

[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. IDEA Publishing Group. 2004 (to appear).

[7] Gasser, L., Penne, B., Transformation of Sensemaking in Networked Organizations. Paper prepared for: First International Conference on the Economic and Social Implications of Information Technology, Washington, D.C. January 27-28, 2003.

[8] Hahn R. (ed.), Government Policy toward Open Source Software. AEI-Brookings Joint Center for Regulatory Studies, Washington D.C. November 2002.

[9] Markus, M.L., Majchrzak, A., Gasser, L., A Design Theory for Systems that Support Emergent Knowledge Processes. MIS Quarterly, 26(3). 2002.

[10] NSF-BRAPC, National Science Foundation Blue Ribbon Advisory Panel on Cyberinfrastructure, Revolutionizing Science and Engineering through Cyber-Infrastructure. February 2003.

[11] Orlikowski, W.J., Knowing in Practice: Enacting a Collective Capability in Distributed Organizing. Organization Science, 13(3), pp 249-273. May-June 2002.

[12] Parnas, D.L., Clements, P.C., A Rational Design Process: How and Why to Fake it. IEEE Trans. Software Engineering, 12(2), pp 251-257. 1986.

[13] PITAC (Presidents Information Technology Advisory Committee), Developing Open Source Software to Advance High End Computing. October 2000.

[14] Ripoche, G., Gasser, L., Scalable Automatic Extraction of Process Models for Understanding F/OSS Bug Repair. To appear in Proceedings of the 16th International Conference on Software & Systems Engineering and their Applications (ICSSEA-03). December 2003.

[15] Scacchi, W., Understanding Software Process Redesign using Modeling, Analysis and Simulation, Software Process--Improvement and Practice. 5(2/3), pp 183-195. 2000.

[16] Scacchi, W., Understanding the Social, Technological, and Policy Implications of Open Source Software Development. Position paper presented at the NSF Workshop on Open Source Software. January 2002.

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

[18] Scacchi, W., Process Models in Software Engineering. In Marciniak, J. (ed.), Encyclopedia of Software Engineering, 2nd Edition, pp 993-1005, Wiley. 2002.

[19] Scacchi, W., Open EC/B: A Case Study in Electronic Commerce and Open Source Software Development. Working paper, Institute for Software Research, UC Irvine. July 2002.

[20] Scacchi, W., Free/Open Source Software Development Practices in the Computer Game Community. Submitted for publication. April 2003.

[21] Scacchi, W., Understanding Free/Open Source Software Evolution: Applying, Breaking and Rethinking the Laws of Software Evolution. Submitted for publication. April 2003.

[22] Scacchi, W., When is Free/Open Source Software Development Faster, Better, and Cheaper than Software Engineering? In Koch, S. (ed.), Free/Open Source Software Development. IDEA Publishing Group, 2004 (to appear).

[23] Truex, D., Baskerville, R., Klein, H., Growing Systems in an Emergent Organization. Communications of the ACM, 42(8), pp 117-123. 1999.

[24] Vixie, P., Software Engineering. In DiBona, C., Ockman, S., Stone, M. (eds.), Open Sources: Voices from the Open Source Revolution. pp 91-100, O'Reilly, Sebastopol, CA. 1999.