

Understanding Continuous Design in F/OSS Projects

Les Gasser^{1,2}, Walt Scacchi², Bryan Penne¹, and Gabriel Ripoché^{1,3}

Graduate School of Library and Information Science¹
University of Illinois, Urbana-Champaign, USA

Institute for Software Research²
University of California, Irvine, USA

LIMSI-CNRS³
Orsay, France

{gasser | bpenne | gripoche @uiuc.edu}
{wscacchi@ics.uci.edu}

Address for correspondence: Prof. Les Gasser, GSLIS/UIUC, 501 East Daniel Street, Champaign, IL, 61820 USA. Tel: +1-217-265-5021; Fax: +1-217-244-3302; Email: gasser@uiuc.edu

Extended Abstract

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. The research process is based on qualitative and computational analysis of large corpuses of large bodies of longitudinal data from Web sites, public repositories, online discussion forums, and online artifacts from F/OSS development projects including 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 5M individual comments, and involving over 60,000 reporters and developers.

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* [Gasser and Penne, 2003] instead of "fixing bugs". 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---constituted the current, and evolving, "specification."

2. Software builders rely on shared collections of "software development informalisms," [Scacchi 2002b], 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 (cf. [Benkler, 2002-03]), 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. 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. What we are seeing is a type of "emergent knowledge process" [Markus et al., 2002].

These observations are significant and surprising. They are significant because this is not a prediction that follows from the current principles of software design, software engineering, and software evolution [Scacchi 2003c] which generally see a sequential (Waterfall model), or an iterative and incremental (Spiral model), design process and development life cycle [Scacchi 2002c]. In contrast, we know that F/OSS are produced in a manner that can be more productive, higher quality, and lower cost than current principles of software engineering suggest [Scacchi 2004].

The observations are surprising because they suggest that even though F/OSS development projects don't rely on explicit representations or formal notations for designs [Pixie 1999], 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 [Elliott and Scacchi 2004, Truex 1999, Scacchi 2002b,d]. Thus these F/OSS projects have no need to "fake" a rational design process, as is all too common in software engineering [Parnas and Clements 1986]. Instead, F/OSS projects seem to rely on enacting a collective capability for distributed design and organizing (cf. [Orlikowski 2002]). How is this possible? That is, how does a globally dispersed community of F/OSS developers design and redesign complex software systems in a continuously emergent manner?

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 [PITAC 2000,

NSF-BRAPC 2003]. E-Government initiatives around the world are increasingly advocating or mandating the use of F/OSS in their system design and development efforts [EGovOS 2003, Hahn 2002]. Third, 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 [Brown and Booch 2002] in ways that can make the design and development of complex software faster, better, and cheaper [Scacchi 2004]. 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 [Scacchi 2002a], or how they might redesign their own processes to assess and take advantage of F/OSS technologies [Scacchi 2000].

Our research approach uses both human-based grounded-theory and automated concept, data, and text-mining methods to empirically discover, identify and comparatively analyze patterns of continuous software design and redesign in the projects under study. The project uses both human qualitative and computational analysis methods to identify design episodes; basic design processes (such as collective sensemaking, negotiation, information seeking, conflict management, uncertainty reduction, evaluation, etc.); design-management activities; and design-support/design-management infrastructure. We analyze the structures of relationships between these elements, including interdependencies, temporal processes, and (mis)fit, and collective underlying social organization. Explanatory models are built by linking patterns of relationships among these elements to outcomes such as ease-of-design-for-modification; persistence vs. resolution of design problems; amount of information exchanged; kinds of skills needed; kind and character of feedback loops; and revisions to the structure of social organization.

As part of this work, we have developed several prototype tools that couple human-based qualitative analysis with computational discovery and analysis methods [Gasser et al., 2003; Gasser, 2003]. For example, we are able to use statistical text-analysis tools to automatically extract episodes of design activity when traces of those activities can be characterized with specific "language models." We have also been able to automatically extract and code 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.

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" [Bodker, 1999].

1. 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 the only way they have of learning about artifacts is indirect. 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 do see the following:

The knowledge development/transformation process is catalyzed by shared, structured, and relatively simple or complex knowledge management environments/tools such as ***Bugzilla***, ***Jitterbug***, ***Bugrat***, ***Gnats***, ***Scarab***, newsgroups, hypermail, etc. that persistently capture traces of design and analysis processes.

The knowledge development/transformation process is characterized by *extreme multiplicity* of viewpoints, representations, experiences, and basic social processes enacted.

The relatively "linear with feedback loops" structure of the more standard software development process is replaced by a network of processes arranged in a highly dynamic topology, including design, construction, testing, releasing, work coordination, critiquing, use, suggesting, specification, tool-building, triage, negotiation, etc. - many more "basic social processes" than those represented in a rational design scenario.

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

Standard models of knowledge representation that underly problem report databases and systems assume a conventional relation between problem reports and discussion (documents) and the use and testing experiences they represent. In this view, *documents* bear a *relationship* to a experiential *life-world*; 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, as follows:

1. ***Documents record viewpoints, and these viewpoints are subject to multiple interpretations.*** Many comments are typically made on each problem report (e.g. an average of 10 comments per report for the 128,000+ reports in our Bugzilla snapshot). These comments often contradict or conflict with each other, and some comments are not interpretable by some readers and respondents. Moreover problem are reported more than once in different ways, and it is often difficult to correlate or link these reports and their underlying manifestations.
2. ***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, both in structure (as the underlying tools develop) and in content (as commentary is added)

3. ***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----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.
4. ***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.

In the end, these results are providing new perspectives on community-wide practices of capturing and managing software design knowledge, how it is articulated, how it breaks down and is reconstituted, and 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.

References

Augustin, L., D. Bressler, and G. Smith, Accelerating Software Development through Collaboration, Proc. 24th. Intern. Conf. Software Engineering, Orlando, FL, 559-563, May 2002.

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>

Bodker S., Computer applications as mediators of design and use - a developmental perspective. Report DAIMI PB-542, Computer Science Department, Aarhus University, October 1999.

Brown, A.W. and G. Booch, Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors, Proc. 7th Intern. Conf. Software Reuse, Lecture Notes in Computer Science, Vol. 2319, 123-136, 2002.

EGovOS, The Center for Open Source & Government, <http://www.egtovos.org>, 2003.

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

Gasser, L., Building and Analyzing PFSA's as Process Representations, SQA Project Memo UIUC-2003-17, 20 May 2003.

Gasser, L. and B. Penne, 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 27-28th, 2003.

Gasser, L., Penne, B., Ripoche, G., and Sandusky, R., Computational Amplification of Qualitative Analysis: Mining Social Processes with Language Models. SQA Project Memo UIUC-2003-14, 25 April, 2003.

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

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

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

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

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

PITAC, Presidents Information Technology Advisory Committee, Developing Open Source Software to Advance High End Computing, October 2000.

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

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 2002a.

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

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

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 2002d.

Scacchi, W., Free/Open Source Software Development Practices in the Computer Game Community, submitted for publication, April 2003a.

Scacchi, W., Understanding Free/Open Source Software Evolution: Applying, Breaking and Rethinking the Laws of Software Evolution, submitted for publication, April 2003b.

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

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

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