

# **Proceedings of the OSS 2010 Doctoral Consortium**

*May 30, 2010*

*Notre Dame, Indiana*

*Collocated with the 6th International Conference on Open Source  
Systems (OSS 2010), May 30–June 2, 2010, Notre Dame, Indiana*

*Edited by:*

***Walt Scacchi***

*University of California*

*USA*

***Kris Ven***

*University of Antwerp*

*Belgium*

***Jan Verelst***

*University of Antwerp*

*Belgium*

**Editors:**

Walt Scacchi  
University of California  
Irvine, USA

Kris Ven  
University of Antwerp  
Antwerp, Belgium

Jan Verelst  
University of Antwerp  
Antwerp, Belgium

ISBN-13: 978-90-337-0034-7

EAN: 9789033700347

Printing and binding: Universitas (Unie van 1991) N.V., Antwerp, Belgium

Published by: Universitas (Unie van 1991) N.V., Antwerp, Belgium

Proceedings copyright © 2010: The aforementioned editors reserve the right to reprint and reproduce the proceedings of the Doctoral Consortium.

Papers copyright © 2010: All papers are copyrighted by their respective author. All authors reserve the full right to copy, reprint or republish their respective paper. Therefore no paper or part thereof may be reproduced without the written permission of the appropriate author.

# Preface

We are pleased to introduce the proceedings of the *OSS 2010 Doctoral Consortium*. The OSS 2010 Doctoral Consortium was collocated with the *6th International Conference on Open Source Systems (OSS 2010)* and was held in Notre Dame, Indiana (USA) on May 30, 2010. The goal of the Doctoral Consortium was to provide PhD students with an environment in which they could share and discuss their goals, methods and results before completing their research. As well, because of the diversity of the communities involved, the Doctoral Consortium allowed PhD students to make connections beyond their own disciplines. The OSS 2010 Doctoral Consortium welcomed the submission of *full papers* and abstracts for the special “*lightning talks*” session.

Part I of the proceedings contains the papers that were accepted as a full paper. Each paper was peer reviewed by at least two independent reviewers. Eventually, we accepted 7 research proposals for inclusion in the proceedings and for presentation at the Doctoral Consortium. During the Doctoral Consortium, PhD students gave a 20-minute presentation of their research proposal and elaborated on the current status of their research. Each presentation was followed by a 20-minute open discussion during which PhD students were provided with feedback on their work from faculty members as well as other PhD students, allowing them to enhance their own research proposal.

Part II of the proceedings contains the abstracts of the presentations given during the lightning talks session. These abstracts were reviewed by the co-chairs of the OSS 2010 Doctoral Consortium based on their relevance to the OSS domain. Based on this screening, 6 abstracts were accepted. During the lightning talks session, PhD students were able to briefly present their research proposal during a 3-minute time slot, followed by a short reaction from the audience. The lightning talks session was primarily targeted towards PhD students who were in the early phases of their research. The lightning talks session allowed PhD students to give a brief presentation of their research, to actively participate in the Doctoral Consortium, and to generate awareness of their topic.

We hope that all PhD students who participated in the OSS 2010 Doctoral Consortium benefited from attending and wish them all the best in the further completion of their research.

June 2010

*Walt Scacchi  
Kris Ven  
Jan Verelst*



## Acknowledgments

We gratefully acknowledge the contributions of several people to the organization of the OSS 2010 Doctoral Consortium. We would like to thank Greg Madey—general chair of the OSS 2010 conference—for his assistance in planning the Doctoral Consortium and for providing logistical support. We further would like to thank the members of our program committee—Kevin Crowston, Joseph Feller, Daniel M. German, Jesus Gonzalez-Barahona, Björn Lundell, and Maha Shaikh—for their valuable input by providing detailed and constructive feedback on the research proposals both during the review process and the Doctoral Consortium. Special thanks go to Kevin Crowston for obtaining financial support from the US National Science Foundation to support travel and attendance for US PhD students to the OSS 2010 Doctoral Consortium and the main OSS 2010 conference. In this regard, we gratefully acknowledge the external financial support provided by the US National Science Foundation under award number IIS-1005183. Thanks to the efforts of all these people, and the PhD students who participated in the event, we were able to make the OSS 2010 Doctoral Consortium a success.

*Walt Scacchi  
Kris Ven  
Jan Verelst*



# Doctoral Consortium Organization

## Doctoral Consortium Co-Chairs

Walt Scacchi	University of California, Irvine	USA
Kris Ven	University of Antwerp	Belgium
Jan Verelst	University of Antwerp	Belgium

## Program Committee

Kevin Crowston	Syracuse University	USA
Joseph Feller	University College Cork	Ireland
Daniel M. German	University of Victoria	Canada
Jesus Gonzalez-Barahona	Universidad Rey Juan Carlos	Spain
Björn Lundell	University of Skövde	Sweden
Maha Shaikh	London School of Economics	UK



# Contents

---

## Part I Full Papers

---

Developers' Contribution to Structural Complexity in Free Software projects . . . 3  
*Antonio Terceiro*

Software Evolution and Human Resources: Knowledge Gap left by Developers  
due to Turnover . . . . . 15  
*Daniel Izquierdo-Cortazar (Advisors: Jesus M. Gonzalez-Barahona and  
Gregorio Robles)*

Evolution of Open Source Software Networks . . . . . 25  
*Matthew Van Antwerp*

Behind Linus's Law: Investigating Creative Peer Review Processes in Open  
Source . . . . . 41  
*Jing Wang*

The Role of Requirements in Open Source IT Innovation . . . . . 53  
*Daniel Curto Millet*

Open Source — Open Community? A Critical Analysis of Stakeholder  
Integration in Free and Open Source Communities . . . . . 65  
*Celina Raffl*

The Impact of Intellectual Property Enforcement on Open Source Software  
Adoption . . . . . 79  
*Wen Wen*

---

## Part II Lightning Talks

---

Classifying Open-source Project Requirements According to McCall's Model  
using a Six-level Tagging Grammar . . . . . 93  
*Radu Vlas*

Incumbent vs. Challengers in the Office Productivity Software Market . . . . . 95  
*Aaron Baird, T. S. Raghu, Rajiv Sinha*

XIV Contents

Improving Open Source Software Patch Contribution Process: Methods and Tools 97  
*Bhuricha Deen Sethanandha*

Moving Beyond Continuance Intention toward Loyalty: An Empirical Study in  
the FLOSS Context . . . . . 99  
*Namjoo Choi*

Impacts of Requirements Engineering Distribution in Open Source Software  
Development Projects . . . . . 101  
*Veeresh Thummadi, Sean Hansen, and Kalle Lyytinen*

Sustaining Processes within the Drupal Open Source Community . . . . . 103  
*Sunah Suh*

Part I

**Full Papers**



# Developers' Contribution to Structural Complexity in Free Software projects

Antonio Terceiro

Computer Science Department  
Federal University of Bahia  
terceiro@dcc.ufba.br

**Abstract.** Free software projects are developed in a way that is substantially different from “conventional” software development. As they get more importance in the Information and Communication Technology ecosystem, understanding the processes by which these projects are developed, evolved and maintained gets more important as well. Structural Complexity, the complexity inherent to the organization of source code elements into modules, represents a major threat to any software project: highly complex software needs higher levels of effort for maintenance activities and exhibit a higher amount of more bugs. Free software projects with highly complex source code are also less likely to attract new developers. This PhD research aims at explaining variations in the Structural Complexity of free software projects in terms of the characteristics of the developers producing it. This paper describes the research by presenting its objectives, proposed research design, and preliminary findings.

**Key words:** Free Software, Open Source Software, Structural Complexity, Core and Periphery, Developer Evolution

## 1 Introduction

The Structural Complexity of a software system is the complexity exhibited by the organization of its modules. It involves both the internal organization of each module, and the relationships between the different modules. Higher Structural Complexity is known to impact negatively programmer productivity in maintenance activities, consequently making it more difficult to fix bugs and add new features. Free software projects<sup>1</sup>, in special those that count only with volunteer developers, are less likely to get new contributions because of that.

The Structural Complexity of a free software project is built one step at a time: the design decisions that shape the code base, making it more or less complex over time, are made by its developers as part of their daily development activities.

This research aims at investigating the factors that influence the increase of Structural Complexity in free software projects. Our hypothesis is that variations in Structural Complexity can be explained by characteristics of the developers, including their

---

<sup>1</sup> In the context of this paper, “Free Software” is used as a synonym for “Open Source Software” (OSS), “Free/Open Source Software” (FOSS) and “Free/Libre/Open Source Software”(FLOSS).

level of participation, experience in the project, experience in specific parts of the project, and the extent to which they are specialised in specifics parts of the project.

The remainder of the paper describes the research, its theoretical basis, goals and current state, and is organized as follows: section 2 presents the theoretical background; section 3 describes the motivation for the research; section 4 presents the research questions; section 5 is about the research design; sections 6 discussed preliminary results, and section 7 finishes the paper by drawing the conclusions.

## 2 Background

This section presents the theoretical background to the research, by exploring two important topics:

- The concept of *Structural Complexity*, its effects on effort needed for software maintenance activities, and studies relating it to other aspects of software projects.
- Different aspects of *developers' participation in free software projects*. We believe that these aspects can explain, at least in part, the variation of Structural Complexity in free software projects. In this paper two different aspects of developer participation are discussed: the core/periphery dichotomy and the developers' evolution.

### 2.1 Structural Complexity

Structural complexity is an architectural concern: it involves both the internal organization of software modules, as well as how these modules relate to each other [6, 3]. Structural Complexity influences the developer's time: a more complex software is expected to require more effort from developers to be comprehended in maintenance activities [10].

Several aspects of Software Design can be considered when evaluating Structural Complexity. We can consider, among others, coupling [5, 11], cohesion [5], and inheritance [5, 11]. While inheritance is specific to the object-oriented paradigm, coupling and cohesion are more generally applicable. Every programming paradigm has a notion of *module*, whether it is called “module”, “class”, “aspect”, “abstract data type”, “source file”, etc. Having modules, one can always analyse a program and identify which other modules a module refers to and thus have a notion of coupling, and also verify how the subparts of a given module interact with each other to evaluate the cohesion of such a module.

In an experimental setting with professional software developers, Darcy *et al* found that more complex software requires more effort for maintenance activities [10]. Moreover, they verified that neither coupling nor lack of cohesion by themselves could explain the decrease in comprehension performance of the developers; only when considered together (by multiplying the two) they presented an association with higher maintenance effort. The authors claim that when considering Structural Complexity, one must consider coupling and cohesion together.

Midha [16] studied projects from sourceforge.net and verified that increases in complexity leads to increase in the number of bugs in the source code, decrease in

contributions from new developers and increase in the time taken to fix bugs. Although using a different concept of Structural Complexity by considering McCabe's Cyclomatic Complexity and Halstead's Effort<sup>2</sup>, these results demonstrate that complexity has nasty effects on Free Software projects. We speculate that an increase in Structural Complexity as studied here has similar effects (although we cannot claim that effectively yet).

Increasing complexity, thus, brings all kinds of trouble to Free Software projects. Stewart *et al* studied 59 projects written in Java that were available on Sourceforge [22], using the product of coupling and lack of cohesion as their Structural Complexity measure. They verified 4 different patterns of Structural Complexity evolution, of which 2 presented growing trend in the end of the period. The other 2 presented stabilization in the end of the period: none of the identified patterns featured a complexity reduction trend. A previous study of ours also indicated a growing trend in Structural Complexity on another (but smaller) project written in C [23].

Increasing complexity trends are not an exclusive feature of free software projects, though: the seminal work of Lehman on software evolution already identified it, and that led to the formulation of the second law of software evolution the Law of Increasing Complexity [14]. That law, formulated in the context of studies on proprietary software systems, states that as systems evolve, their complexity increases unless work is done to maintain or reduce it.

For now, we know that i) software complexity is associated with undesirable effects (more maintenance effort, more bugs, less attraction of new developers) and ii) Structural Complexity tends to not decrease, and in a reasonably large amount of cases, it tends to grow. That leads us to the following question: why does Structural Complexity increase in the context of Free Software projects?

Having a more open governance structure seems to be related to better design quality. [4] From one side, a higher design quality enables a more open governance: less coupled modules allow different developers to work on their own parts of the project without explicit coordination activities. Having a more open governance gives developers more freedom to enhance the design quality instead of having to keep up with deadlines or another types of pressures from higher management or customers [4].

Another possible reason for a worse software design quality is probably bad news for most project leaders. Project success<sup>3</sup> may be associated with a lower design quality. When a project reaches a leading position, it may be that the lead developers start to focus on lateral activities rather than on programming, such as answering users in forums or mailing lists, reviewing contributions etc [1].

## 2.2 Developer's participation in free software projects

---

<sup>2</sup> These two measures represent respectively the internal complexity of subroutines and the overall vocabulary size of the code base. They reflect, thus, a different aspect of Structural Complexity. Here we are looking at Structural Complexity at the design/architecture field, considering the relationship between modules and between the sub-parts of each module.

<sup>3</sup> measured as a function of the number of downloads, web traffic and development activity

**The core/periphery dichotomy.** Normally, a Free Software project is started by a single developer, or by a group of developers, in need of addressing a particular need. After there is a usable version, it is released to the public under a Free Software license which allows anyone to use, change and distribute a copy of that software. As new users get interested in the project, some of them may start to contribute to it in several possible ways: with code for new features or bug fixes, with translations into their native languages, with documentation, or with other types of contribution. At some point, then, the project has a vivid and active community: a group of people that gravitate around a project, with varied levels of involvement and contribution.

The “onion model” [8, 17] became a widely accepted representation of what happens in a Free Software project, by indicating the existence of concentric levels of contribution: a small group of core developers do the largest part of the work; a larger group makes direct, but less frequent contributions in the form of bug fixes, patches, documentation, etc; an even larger group reports problems as they use the software, and the largest group is formed by the silent users who only use the software but never provide any type of feedback.

The processes by which participants migrate from one group to another are very different from one community to the other: communities may adopt more formal and explicit procedures for that, or use a more relaxed approach and let things flow “naturally”. But in general the achievement of central roles (and thus more responsibility, respect and decision power) are merit-based: a developer becomes a leader by means of continuous valuable contributions to the community [13].

Since most of the work is done by a core team, it is important for projects to keep a healthy and active core team. Some projects are able to keep its core team with few or no changes across its entire history, while others experience a succession of different generations of core developers [19, 18].

The relationship between core contributors and peripheral (non-core) members of a community are not always smooth: sometimes the core tends to work on their own demands and to give little attention or even to ignore completely the demands of the periphery [9, 15]. From an individual point of view, core and periphery members also exhibit different behaviour while debating subjects related to the project [21] or in the bug reporting activity [15].

**Developer Evolution.** There is a large number of studies on the evolution of Free Software projects. Most of them are concerned with the evolution in the projects’ internal attributes, such as size and to some extent software architecture.

In order to understand software evolution of Free Software projects, however, one needs to understand the evolution of their communities [20]. This understanding must comprehend not only the growth of communities [25], but needs to take the evolution of individual developers into account as well.

By analysing developer’s activity in a Free Software project, we can identify several processes they go through in the course of their evolution: [7]

- Some developers remain working in the same set of modules and are *specialists*, while others, *generalists*, achieve a broader experience in the project and change a growing number of modules.

- Developers achieve different centrality measures in the project when considering the modules they change: some happen to change the project's more central — and important — modules, while others change only non-central modules.
- Developers shift from the periphery to the core of the project, or vice-versa.

Understanding the process of developer evolution in Free Software projects is an important step towards understanding the projects's own evolution.

### 3 Motivation for the Research

From times to times, we hear stories about free software projects being rewritten from scratch. Just to cite recent examples, both `eog`, GNOME's image viewer and `gnome-session`, GNOME's session management software, got completely rewritten into new versions<sup>4</sup>. In these projects, the code base became so difficult to maintain that the active developer(s) decided that rewriting them was worth, given the high amount of effort required to keep maintaining them in their current state. Better, less complex code would make maintenance easier by requiring less effort to add new features and fix bugs in a way that does not make future maintenance harder.

When such a rewriting effort is made, precious developer effort is spent in a complete redesign of the software, instead of on implementing new features and fixing bugs; every project leader would prefer not having to do it. If we can identify which factors contribute to added complexity in free software projects we'll be able to avoid having projects reaching the point in which its developers start to consider a complete rewrite.

As seen in section 2.1, there is a theoretical construct that characterises the problem: Structural Complexity. Software with high Structural Complexity is harder to maintain and evolve, has more bugs, and is less likely to attract new contributors.

Although we know what consequences higher levels of Structural Complexity can have in projects, we have little knowledge about its causes. If we understand the factors that affect the increase (or decrease) in Structural Complexity in free software projects, project leaders will be able to deploy methods and techniques to mitigate these factors and thus avoid having an unacceptable level of complexity in their source code. This will make it easier for the project to attract contributors as well as avoiding the need for a complete rewrite.

In this research we explore developer characteristics as factors that may influence the variation of Structural Complexity in free software projects.

### 4 Research Questions

The *general goal* of the research is to build a model of how developers influence the evolution of Structural Complexity in Free Software projects. This includes identifying

---

<sup>4</sup> These rewrites were described in their wiki pages, respectively <http://live.gnome.org/EyeOfGnome/EogNg> and <http://live.gnome.org/SessionManagement/NewGnomeSession>.

factors related to characteristics of the developers that influence Structural Complexity, as well as reporting the conditions under which such influence is observed. Our hypothesis is that the Structural Complexity variation in a Free Software project can be explained by characteristics of the developers that change its source code.

Specifically, we are working on the following *research questions*:

1. Does the *developers' level of participation* affect Structural Complexity? This involves verifying whether core and peripheral developers introduce different amounts of Structural Complexity in the source code.
2. Does *individual developers' experience in the project* affect Structural Complexity? This involves investigating whether the amount of Structural Complexity added by a developer changes as he/she evolves in the project.
3. Does *individual developers' experience in specific parts of the project* affect Structural Complexity? We will investigate whether developers introduce different amounts of Structural Complexity when changing modules they are used to in comparison with when they change modules they are not so used to.
4. Does *specialisation and generalism* affect Structural Complexity? Do specialist developers introduce different amounts of Structural Complexity with their changes in comparison with generalist developers?

Providing answers to these questions will provide original contribution to the body of knowledge about the Free Software development phenomenon, and to some extent to the wider Software Engineering community as well.

## 5 Research Design

The overall research definition, using a GQM template [2], is as follows: in this research we *analyse* changes made to the source code of free software projects as stored in their version control repositories *with the goal* of characterization *with respect* to structural complexity added or removed, level of developer engagement, developer experience in the project, developer experience with the modules changed and developer specialisation *from the perspective* of the researcher *in the context of* free software projects.

Figure 1 shows a model of the research following the GQM paradigm. Our main goal is identifying the factors that influence the variation of Structural Complexity in free software projects, what is shown in the first column. This goal unfolds itself into the four research questions in the second column. The research questions are associated with the metrics (or variables) in the third column.

Each research question will be investigated in a empirical study, in which the corresponding variables will be analysed looking for correlation relationships. Further theoretical work will be done in order to identify cause-effect relationships as well.

### 5.1 Variables and Operational Definitions

This section describes the variables considered in the research. They are measured with respect to each change extracted from the projects' version control repositories.

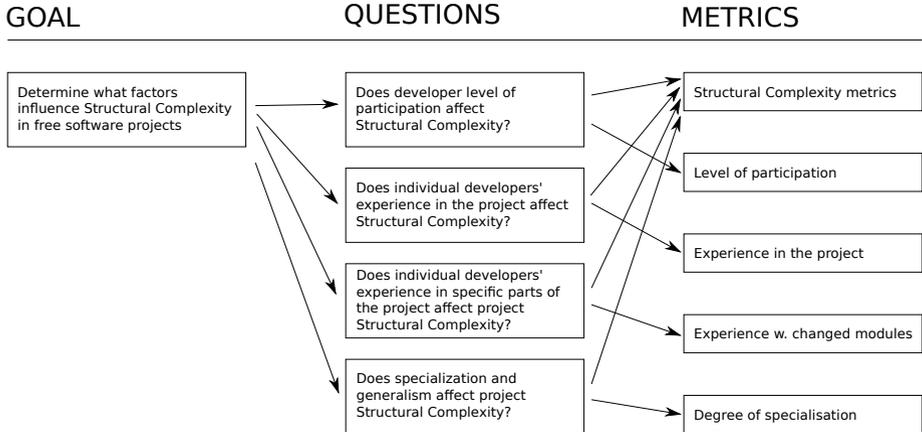


Fig. 1. GQM model of the research.

The independent variables are related to the factors investigated in each research question, and the dependent variables are related to the concept of Structural Complexity as considered in this research. The variables are also presented in the GQM model of the research (figure 1), in which they are related to the corresponding research questions. While in the GQM model the Structural Complexity variables are grouped together as “Structural Complexity metrics”, below they are described individually for completeness.

### Independent variables

- *Level of participation* —  $L$ . This variable represents whether the change was made by a core or a peripheral developer. To determine the value of this variable, first the analysed period is split into 20 periods of equal duration. Each change is then considered as being made by a core developer if its author is one of the 20% top committers in the corresponding period, or as being made by a peripheral developer otherwise (cf. [19, 18]). Using this definition, one should note that the same developer can be considered as a core developer in some periods and a peripheral developer in other periods. This is coherent with reality: developers may reduce or increase their activity their involvement in the project in specific periods.
- *Experience in the project* —  $E_p$ . The number of previous changes made by the same developer in the project as a whole
- *Experience with the modules being changed* —  $E_m$ . Number of previous changes by the same developer that affected the modules being changed. If more than one module is being changed, use the average value of all modules.
- *Degree of Specialisation* —  $S$ . We calculate the ratio between the number of modules the user changed previously and the total number of modules in the project at the time of a change. Specialist developers will have this ratio close to 0 and generalist developers will have it closer to 1. By subtracting this ratio from 1, we have a degree

of specialisation that goes from 0 (generalist developer) to 1 (completely specialist developer).

### Dependent variables

- *Overall Structural Complexity* —  $SC$ . This variable represents the overall Structural Complexity of the project after each change and is obtained by multiplying average coupling and average lack of cohesion metrics: since coupling and cohesion are normally module-level metrics, we take the average of all modules to have a project-level value.
- *Variation in Structural Complexity* —  $\Delta SC$ . This is the increment in Structural Complexity caused by each change. For each change, this value is obtained by subtracting its  $SC$  value from the  $SC$  value of its previous change. This variable represents how much the Structural Complexity changed after a given change was applied to the project source code.
- *Absolute variation in Structural Complexity* —  $|\Delta SC|$ . The absolute change in Structural Complexity. It's the absolute value of  $\Delta SC$ . When restricting the analysis to only those changes with positive or negative  $\Delta SC$ , this value can be used as a measure, respectively, of how much Structural Complexity has increased or reduced.

## 5.2 Data sample and collection approach

We plan to select free software projects from 3 or 4 application domains according to the following criteria:

- *Available in Debian GNU/Linux*. This is considered as an indication that the software in question is actually used: if someone cares enough about the project to package and maintain it for easy installation by other users, than we consider that the project in question is minimally relevant.
- *Written in C, C++ or Java*. The source code analysis tool we are using was only sufficiently tested with such languages.
- *Publicly accessible version control repository*. This is necessary so that we can obtain the data from the version control repositories.

The source code repository of each project is imported locally in a `git`<sup>5</sup> repository to facilitate fast and off-line history browsing. We then use a set of scripts developed by us to mine this repository as follows:

- Determine the list of relevant commits, by identifying the commits that changed source code files. This way we avoided analysing subsequent states of the source code that were no different from each other.
- Checkout each relevant version and run a static source code analysis tool to calculate the source code metrics used, namely CBO [5] and LCOM [5] (we used the improved version from Hitz and Montazeri [12], though).

---

<sup>5</sup> <http://git-scm.org/>. `git` has support for importing repositories from CVS and Subversion.

- Extract the information needed to calculate the other variables, such as modules touched in the change, the name and e-mail of developer who made the change, date of the change etc.
- Accumulate the results for each change in a single data file per project.

After processing all the projects, their raw data is loaded in a relational database in order to facilitate the calculations of the variables defined in the research design.

## 6 Preliminary Results

A first empirical study, addressing research question 1, was already performed and is currently under review for publication [24]. The study consisted of a field experiment in which data was collected from the version control repositories of 7 web server projects written in C, and compared the amounts of Structural Complexity introduced by core and peripheral developers. We have found that in general the changes made by core developers introduce less Structural Complexity than the changes made by peripheral developers. When the analysis was restricted to the changes that reduced Structural Complexity, the ones made by core developers accomplished a larger Structural Complexity reduction than those made by peripheral developers. These results demonstrate the importance of having a stable and healthy core team to the sustainability of free software projects.

Ongoing work includes verifying the relationship between Structural Complexity and the variables related to research questions 2, 3 and 4 .

## 7 Conclusions

This research aims to provide an understanding about the relationship between developer characteristics and the introduction of Structural Complexity in the source code of free software projects. We argue that the variation in Structural Complexity can be explained by evaluating attributes of the developers that change the projects' source code.

Expected contributions include extending the knowledge currently available about software quality issues in free software projects. By identifying the factors that affect the increase of Structural Complexity in free software projects, we will help projects leaders to employ strategies that keep complexity in their projects at an acceptable level, this way avoiding increase in bugs, enabling the involvement of new contributors and easing the project's maintenance and evolution.

Preliminary results indicate that core and peripheral developers introduce different levels of Structural Complexity in the source code, and that core developers have a higher impact in complexity-reducing activities. That indicates the importance of the core team in Free Software projects.

Further work will evaluate developer experience in the project, developer experience with specific modules and developer degree of specialization in the project as influential factors in the evolution of Structural Complexity.

## References

1. Donato Barbagallo, Chiara Francalenei, and Francesco Merlo. The Impact of Social Networking on Software Design Quality and Development Effort in Open Source Projects. In *ICIS 2008 Proceedings*, 2008.
2. Victor Basili, Gianluigi Caldiera, and Dieter H. Rombach. The Goal Question Metric Approach. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley, 1994.
3. Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
4. E. Capra, C. Francalanci, and F. Merlo. An Empirical Study on the Relationship Between Software Design Quality, Development Effort and Governance in Open Source Projects. *IEEE Transactions on Software Engineering*, 34(6):765–782, Nov.-Dec. 2008.
5. S.R. Chidamber and C.F. Kemerer. A metrics Suite for Object Oriented Design. *IEEE Trans. Software Eng.*, 20(8):476–493, 1994.
6. Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting software architecture : views and beyond*. The SEI series in software engineering. Addison-Wesley, Boston, 2002.
7. Jean M. dos R. Costa, Francisco W. Santana, and Cleidson R. B. de Souza. Understanding Open Source Developers’ Evolution Using TransFlow. In *Groupware: Design, Implementation, and Use, 15th International Workshop, CRIWG 2009, Peso da Régua, Douro, Portugal, September 13-17, 2009. Proceedings*, pages 65–78, 2009.
8. Kevin Crowston and James Howison. The Social Structure of Free and Open Source Software Development. *First Monday*, 10(2), 2005.
9. Jean-Michel Dalle, Matthijs den Besten, and H ela Masmoudi. Channeling Firefox Developers: Mom and Dad Aren’t Happy. In Barbara Russo, Ernesto Damiani, Scott A. Hissam, Bj orn Lundell, and Giancarlo Succi, editors, *Open Source Development, Communities and Quality, IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software, OSS 2008, September 7-10, 2008, Milano, Italy*, volume 275, pages 265–271. Springer, 2008.
10. D. P. Darcy, C. F. Kemerer, S. A. Slaughter, and J. E. Tomayko. The Structural Complexity of Software: An Experimental Test. *IEEE Transactions on Software Engineering*, 31(11):982–995, Nov. 2005.
11. F. Brito e Abreu. The MOOD Metrics Set. In *Proc. ECOOP Workshop Metrics*, 1995.
12. M. Hitz and B. Montazeri. Measuring coupling and cohesion in object-oriented systems. In *Proceedings of the International Symposium on Applied Corporate Computing*, 1995.
13. Chris Jensen and Walt Scacchi. Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study. In *ICSE ’07: Proceedings of the 29th international conference on Software Engineering*, pages 364–374, Washington, DC, USA, 2007. IEEE Computer Society.
14. M. M. Lehman, J. F. Ramil, P. D. Wernick, and D. E. Perry. Metrics and Laws of Software Evolution-The Nineties View. In *Proceedings of the 4th International Symposium on Software Metrics*, 1997.
15. H ela Masmoudi, Matthijs den Besten, Claude de Loupy, and Jean-Michel Dalle. “Peeling the Onion”: The Words and Actions that Distinguish Core from Periphery in Bug Reports and How Core and Periphery Interact Together. In Cornelia Boldyreff, Kevin Crowston, Bj orn Lundell, and Anthony I. Wasserman, editors, *OSS: Diverse Communities Interacting, 5th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2009, Sk ovde, Sweden, June 3-6, 2009. Proceedings*, volume 299, pages 284–297. Springer, 2009.

16. Vishal Midha. Does Complexity Matter? The Impact of Change in Structural Complexity on Software Maintenance and New Developers' Contributions in Open Source Software. In *ICIS 2008 Proceedings*, 2008.
17. Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.
18. G. Robles, J. M. Gonzalez-Barahona, and I. Herraiz. Evolution of the core team of developers in libre software projects. In *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, pages 167–170, May 2009.
19. Gregorio Robles and Jesus Gonzalez-Barahona. Contributor Turnover in Libre Software Projects. *Open Source Systems*, pages 273–286, 2006.
20. Walt Scacchi. Understanding Open Source Software Evolution. In Nazim H. Madhavji, Juan C. Fernández-Ramil, and Dewayne E. Perry, editors, *Software Evolution and Feedback: Theory and Practice*, chapter 9. John Wiley & Sons, 2006.
21. Michael J. Scialdone, Na Li, Robert Heckman, and Kevin Crowston. Group Maintenance Behaviors of Core and Peripheral Members of Free/Libre Open Source Software Teams. In Cornelia Boldyreff, Kevin Crowston, Björn Lundell, and Anthony I. Wasserman, editors, *OSS: Diverse Communities Interacting, 5th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2009, Skövde, Sweden, June 3-6, 2009. Proceedings*, volume 299, pages 298–309. Springer, 2009.
22. Katherine J. Stewart, David P. Darcy, and Sherae L. Daniel. Opportunities and Challenges Applying Functional Data Analysis to the Study of Open Source Software Evolution. *Statistical Science*, 21:167, 2006.
23. Antonio Terceiro and Christina Chavez. Structural Complexity Evolution in Free Software Projects: A Case Study. In Muhammad Ali Babar, Björn Lundell, and Frank van der Linden, editors, *QACOS-OSSPL 2009: Proceedings of the Joint Workshop on Quality and Architectural Concerns in Open Source Software (QACOS) and Open Source Software and Product Lines (OSSPL)*, 2009.
24. Antonio Terceiro, Luiz Romário Rios, and Christina Chavez. An Empirical Study on the Structural Complexity introduced by Core and Peripheral Developers in Free Software projects, 2010. *To appear*.
25. Yi Wang, Defeng Guo, and Huihui Shi. Measuring the evolution of open source software systems with their communities. *SIGSOFT Softw. Eng. Notes*, 32(6):7, 2007.



# Software Evolution and Human Resources: Knowledge Gap left by Developers due to Turnover

Daniel Izquierdo-Cortazar

(Advisors: Jesus M. Gonzalez-Barahona and Gregorio Robles)

GSyC/LibreSoft, Universidad Rey Juan Carlos, Mostoles, Madrid  
{dizquierdo,jgb,grex}@libresoft.es

**Abstract.** From a traditional point of view, software evolution is a field of research which is commonly interested in how the source code evolves. However, developing software is a task full of intellectual effort and people behind a software are key factors for its evolution. Thus, human resources are also interesting in the field of software evolution or maintenance since the wrong people in the wrong place may provoke chaos. Generally speaking, not everybody has the right to modify the source code, but we may find situations where there is a turnover of developers and senior developers leave a project. In this case, a new developer will take control of the old source code. From an intuitive point of view, this produces a decrease in the level of quality for maintenance activities. Fixing a bug will take more time to a junior developer than to the senior one. Hence, situations such as: discovering areas with no activity during a long time, detecting that the current developer team has a low knowledge of the source code or studying the huge "knowledge gap" left by a developer when she left; are essential for the leaders of a libre software community. This thesis aims to clarify those aforementioned situations from an empirical point of view looking for risky situations due to developer turnover.

**Key words:** Libre software, turnover, data mining, software evolution, human resources, libre software communities

## 1 Introduction

There exist several approaches and metrics related to libre software, both from the proprietary world [6] and from the libre software world. However, libre software consists of several pieces such as communities, publicly available repositories and so on, which allow to better understand their evolution [17],[13], [10].

Specifically, this is the case of libre software communities which are formed by a set of people with same interests and, generally speaking, a common goal and point of view about how to deal with a given problem [7],[1].

In fact, libre software is studied from several points of view, from the traditional software evolution point of view [9],[12], knowledge management, social networks, and so many fields that it is hard to name all of them. This explosion of knowledge creation is based on the publicly available data sources, where most of the data can be found just with a simple click of the mouse [18].

As a brief summary, source code management systems (SCM), mailing lists, bug tracking systems (BTS) and source code releases are the repositories where most of the information can be found.

- Source Code Management System: The best known in libre software world are CVS, Subversion and Git.
- Mailing lists: Most of the times, there are several mailing lists which contain the community discussion.
- Bug Tracking System: This is the issue tracking system where errors or patches are reported in order to be fixed or applied by some developers with commits rights.

## 2 Software Development and Human Resources

Software development is an activity intense in human resources. The work of many developers is required to create almost any non-trivial piece of code. The lifespan of a software system can range from several years to decades. In such scenarios, the development team in charge of the software may suffer from turnover: old developers leave while new developers join the project. With the abandonment of old, senior developers, projects lose human resources experienced both with the details of the software system and with the organizational and cultural circumstances of the project. New developers will need some time to become familiar with both issues.

It is estimated in at least six months to get some progress and experience in performing at the same rate as an old member of the development team [4]. Even more when the development team is working on some more sophisticated work. Unfortunately, experts in specific fields are not really known by their abundance. It means that when some of your experts, as a project manager, leave the project, you may face new problems.

Although maintaining the current development team could be thought as a plausible solution to mitigate this problem, turnover is usually unavoidable. Being a highly intellectual work, developers have a tendency to lose the original motivation on the software system as time passes by, and they have the natural desire to search for new objectives. In this sense, high turnovers have been observed in most large FLOSS projects, where several *generations* of successive development teams have been identified [17]. Although these environments are partially, if not mostly, driven by volunteers, turnover in industrial environments is also high.

Expenses in the project probably are the same as above if you hire another expert, however her expertise is not the same, just because we were talking about team, domain or bureaucracy context. Thus, it is almost impossible to find a new developer as skillful as the last one and the new one will need some adaptation process.

From the companies point of view, when acquiring software, enterprises are not only interested to know about the product and its quality but also interested in who produced that product and its reputability. For traditional enterprises, reputability can be check based on financial strength of the software provider however, for the FLOSS world, it must be found other ways to determine if a FLOSS project is serious. This

can be done by studying the behavior of a FLOSS community. In particular, a FLOSS community should behave in a manner to convince potential FLOSS integrators from Industry that it is dependable.

Thus, determining risky situations is crucial for the enterprises interests and it is not only possible analyzing the source code. It is even more important the set of people working behind it.

The human team in charge of the maintenance process should also be considered to characterize the "health" of a system. A maintainer who has some knowledge about a system will maintain it better than somebody with no experience with it. Hence, we can expect that developers who introduced code which still persists in the system have a certain advantage when maintaining it. The opposite is also true: losing a developer with a large experience contributing to the project, (i.e. who has done many changes), will affect the maintainability and aging of the software.

### 3 Research Question

As it was said, there is a natural regeneration of developers and a given developer team in a given period has certain accumulated knowledge about the current state of the project. For instance, the bureaucratic structure of the project, problem's domain, current status, future evolution or weaknesses and strengths. This is named in some fields as *tacit* and *explicit* knowledge [8] and it remains in somehow in the minds of developers, but also in the aforementioned data sources.

Thus, on the one hand, we can find communities where there is a full regeneration of developers and the current developer's team do not have a real knowledge of the source code. On the other hand, there may be communities with zero regeneration of developers what means that the very first committers of the community are the current ones, even when months or years has gone.

In the first case, where there is an extreme turnover, the developers coming to the community are not familiarized with the source code and new changes may introduce new bugs and also there is, from an intuitive point of view, a tendency to spend more time fixing a bug since there is not a real knowledge about the source code, but also, why some specific source code lines were introduced there, some tricky functions or even some decay in the architecture. In the second case, there is an implicit risk situation in a community where just some of the developers work on the project in an active way, and it will become dangerous when one of them leaves the community. After this, a big quantity of source code managed by a committer with months or years of experienced work is now left to the community and the other developers have to deal with it. However if there are not new developers coming or regenerating the core team, it means more work for the same team and even more, it means more work in not previously known areas.

Once here, there are some hypotheses which are thrown to be answered in this dissertation.

- General Hypothesis check the two extremes to be analyzed:

- **Hypotheses 0:** A high regeneration of developers directly increase the knowledge gap in the project.  
*Threats to validity: We should check core developer team and non core development team, in the second case this is false. But anyway, if some developer leaves the project, the quantity of orphaned code will increase.*
- **Hypotheses 1:** A low regeneration of developers will show a constant decrease of the knowledge gap in the project.  
*Threats to validity: At least the kernel of the application will remain authored by active committers.*
- Specific hypothesis dealing with how the community deals with the problem of knowledge gap.
  - **Hypotheses 2:** The knowledge gap tends to be absorbed by other developers. *Basically it tries to measure the quantity of orphaned code which become authored by new developers. Thus, it tries to check if there is a natural decrease in the number of orphaned areas*
  - **Hypotheses 3:** Communities with large areas of orphaned code (a huge knowledge gap) tend to have more bug reports, these tend to take more time to be fixed, more time to be picked up and there are more tossed bugs because it is more difficult to deal with not known source code. *Some preliminary studies have shown that potential abandoned code shows an increase in the number of bug reports [14]. This hypotheses could be analyzed from the quantitative perspective (directly analyzing data from BTSs, or from the prediction perspective, analyzing source code from the SCMs)*
  - **Hypotheses 4:** Communities with a high level of knowledge gap may deal with code cloning. *For instance checking source code from Evolution (due to its huge knowledge gap) or JetSpeed 1 and JetSpeed 2 (the first one was abandoned and some parts of its source code were taken to start what is a new project: JetSpeed 2).*

## 4 Methodology

This section aims to explain the steps followed to present the preliminary results which can be found in section 5. However, some concepts have not been clearly defined:

- **Committer:** This is the person who has rights to commit a change into the source code.
- **Author:** A commit could be committed by a given committer, but she may not be the real author. Some SCMs offers this information (for instance, Git provides a specific field for this).
- **Orphaned Code:** The orphaned code is the source code whose author has left the project. It means that she was mainly committing it for a given period. This orphaned code could be measured in number of lines, methods or files. It is worth to mention that there are differences between "orphaned code" and "abandoned code". The latter could still have an author working on the project, but it could have not been touched for a long time.

- **Knowledge gap:** We could assume that committers have *complete knowledge* over the lines they commit since, even when she is not necessarily the author, committer have a similar capacity to understand, modify and explain the introduce lines as if they were the actual authors. Thus, if a committer leaves a project, there exist a knowledge gap.

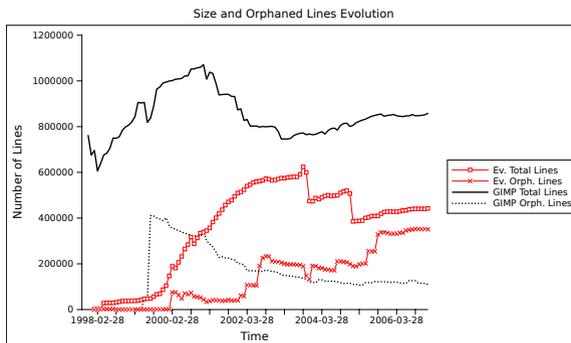
The methodology considers projects that store source code in a SCM, such as CVS, SVN or Git. Thus the whole life of a project can be analyzed. Thanks to the metadata stored in the SCM, it is possible to know when a change took place, who committed that change and what was added, removed or modified. Even more, using the *annotate* command of the SCM we are able to obtain data line by line. This analysis considers source code files. Thus, documentation, translation or multimedia files are ignored.

Hence, once we understand the whole process or changes in the SCM and how to retrieve them, the evolutive analysis is made by means of retrieving, once more, information from the resultant database. Having this database, we know when some developers left a project, and thus, when their authored lines become orphaned lines.

As indicated in [11] the tool used was "Carnarvon"<sup>1</sup>. but we realized that it was too complex in time (e.g.: two months for a GCC monthly analysis). Thus, we have started a new one, providing revision granularity instead of monthly.

## 5 Preliminary Results

Some initial results, presented at HICSS 2009 [11] have shown strong differences in the knowledge gap left by senior developers due to a high turnover. Communities such as Evolution (the email client from GNOME Desktop) has reached a 80% of this knowledge gap, while communities such as GIMP (the image manipulator from GNOME Desktop) shown knowledge gap of around a 15%. This can be shown in picture 1



**Fig. 1.** Evolution of total size and number of *orphaned* lines for Evolution and GIMP.

<sup>1</sup> <http://forge.morfeo-project.org/projects/libresoft-tools/>

Similar results have been found using a granularity of files. It raises another question and it is related to the importance of the granularity. It is not clear enough which is the best granularity. Using a granularity of lines we are deepening in a SLOC study, but ignoring some problems other problems, such as architecture decay or method understanding (what is more usual than working at the level of lines).

## 6 Theoretical Framework: Discussion

This section aims to explain the theoretical framework used in this dissertation. The staged model defined by Vaclav and Bennet [16] shows some interesting concepts which are directly related to the concept of "orphaned lines" and "knowledge gap".

Next, there is a small definition of the phases defined:

- **Initial Development** The system is built from scratch to follow all the initial requirements given by the client. Developers will retrieve most of their knowledge that will be crucial for the correct development of the product.
- **Evolution** The second phase consists of several iterations on the functionality where developers must fix bugs and add, modify or remove source code regions in order to fit with old and new requirements. The fault rate has decreased since there is needed a stable version of the tool. This is the main phase where software evolution takes place and several releases are born.
- **Servicing** In order to evolve the system must have an appropriate architecture and a development team skilled. When one of those concepts is missing, the software enters in the servicing phase and it starts to get old, or in other words, code decays, ages or it is considered as legacy. The architecture starts to miss consistency, developers are moved to some other projects and new changes are becoming more and more difficult.
- **Phaseout.** No more service is offered and the company just looks for as much benefits as possible.
- **Closedown** The product is removed from the market.

Thus, there are main differences among stages, but in all of them the staff expertise is vital to achieve a successful project. This expertise is crucial during the first two stages where developers study the application domain and absorb all the project idiosyncrasy (requirements, iterations, modifications to the architecture and so on. This expertise is totally necessary to implement new functionalities, fix bugs and other maintenance activities. However in the last stages of the project this expertise is not so important because the company is just interested in earn as much money as possible and it implies to move resources from one project to a new one.

The Servicing phase is known as the stage where the architecture starts to miss consistency and resources are being moved from one project to another one. As it was said, during the initial phases developers obtain a high level of knowledge about the source code, the architecture, the development team and other project areas. However in this phase, the project starts to miss senior developers even when the architecture is decaying and those initial developers are being replaced by developers with less

expertise in the domain, in the architecture and, generally speaking in all areas. From an intuitive point of view, a bug will be probably fixed earlier by a senior developer than by a new developer that has just arrived. Six months is the common time to start to be productive, so the regeneration of developers in a project, if it is not well managed, could finish in a risky situation from the project.

So, we are playing with two main concepts, human resources and software decay [15], [5], [2]. Both are directly interrelated since if there is a lack of quality in human resources the source code may start a decay or an aging in some modules or functionalities. On the other hand, the no existence of software decay and good maintenance activities show a good management of the human resources and the regeneration of developers. Both exist and projects must deal with them in order to success.

Capiluppi [3] derived a new staged model for open source where differences from proprietary software and open source software are specified. The research question addressed was if the staged model is suitable for open source projects. In fact, it was, but with some differences between the proprietary software and the open source software:

- Projects driven by the company, employees are paid to carry out the project.
- They have to achieve specific requirements what they are paid for.
- Once they are in the phaseout stage, there is no way to come back and there is a developers turnover.
- When the project is removed from the market, there is no way to get it again, the company has the right to do it.

And on the other hand open source systems:

- Projects are driven by volunteers
- The source code is publicly available, what means that there may appear new developers leading the project, forks, or other situations derived of it.
- There are risky situations when developers leave the project, in most of the cases, there is no money to pay new developers.
- The product remains on the Internet, anyone can take it again, once the project is abandoned.

Based on the aforementioned differences, open source projects show a situation where most of them are driven by volunteers and this is perceived by companies as a risky way of developing. This perception is provoked by the way libre software projects work, there no exist a company behind them, so the way of developing is different from the traditional way. This dissertation tries to come to reality by means of data mining and an empirical software evolution approach to all people interested in checking risky situations on the source code.

As aforementioned, developers in libre software communities may leave the project in a given date, but new developers may not appear to fill this gap. Thus, there is a higher risk of not maintenance activities in certain regions of the source code. There is a research question could be if a high regeneration of developers may produce risky situations for the projects due to the appearance of knowledge gaps. If so, then we want to be able to detect those situations by analyzing the source code management systems

(data mining approach) and studying the life span of developers, their activity in the source code, the knowledge gap left and how it is absorbed by the new members of the community.

## 7 Further Work

Some hypotheses were addressed in section 3, however there is still a necessity to check what that knowledge gap detected in some projects means. From an intuitive point of view, areas with no knowledge and no activity from the current development team may mean areas with no maintenance. However, this is not bad by definition since some of those areas may be mature enough and no activity is needed.

Thus, it is a key factor to guess what that knowledge gap (whatever the granularity is: SLOC, files or functions/methods) means and how this is related to some other activities such as the bug fixing process and the maintenance process.

One of the main research lines proposed in this paper is that there exist a natural regeneration of developers and this is demonstrated by other authors that this affect the maintenance of a project. Even more, libre software projects are mostly driven by volunteers what makes more difficult to fill the gap left by others than compared to projects lead by companies. Hence, there is an inherent risk related to the turnover and it is, from our point of view, identified in those knowledge gap found in some specific projects.

In this circumstances, it is necessary to add extra information from other data sources such as bug tracking systems. These new set of data sources may help to check if potential non-maintained areas show a decrease in maintenance activity. For instance, bugs tend to remain open more time than before. Or bugs from specific areas tend to have less chances to be assigned.

It is also remarkable the fact that the data mining process is not simple and the migrations done by some communities in their repositories could produce "odd" results. Thus, it is also necessary to previously answer the question of what is the best way to retrieve the data needed for this study and how to deal with the main repositories found in libre software projects. For instance, it is not the same to analyze a CVS repository than a Git repository and the idiosyncrasy found in both development models (centralized the first one and fully distributed the second one) may show different results.

## 8 Acknowledgements

This work has been funded in part by the European Commission, under the FLOSS-METRICS (FP6-IST-5-033547), QUALOSS (FP6-IST-5-033547) and QUALIPSO (FP6-IST-034763) projects, and by the Spanish CICyT, project SobreSalto (TIN2007-66172).

## References

1. J. Bacon. *The Art of Community: Building the New Age of Participation*. O'Reilly Media, Inc., 2009.
2. A. Capiluppi and K. Beecher. Structural complexity and decay in floss systems: An inter-repository study. *Software Maintenance and Reengineering, European Conference on*, 0:169–178, 2009.
3. A. Capiluppi, J. M. González-Barahona, I. Herraiz, and G. Robles. Adapting the "staged model for software evolution" to free/libre/open source software. In *IWPSE '07: Ninth international workshop on Principles of software evolution*, pages 79–82, New York, NY, USA, 2007. ACM.
4. T. De Marco and T. Lister. *Peopleware : Productive Projects and Teams, 2nd Ed*. Dorset House Publishing Company, Incorporated, 1999.
5. S. Eick, T. Graves, A. Karr, J. Marron, and A. Mockus. Does code decay? assessing the evidence from change management data. *Software Engineering, IEEE Transactions on*, 27(1):1–12, Jan 2001.
6. N. Fenton. *Software Metrics: A Rigorous Approach*. Chapman and Hall, 1991.
7. K. Fogel. *Producing Open Source Software : How to Run a Successful Free Software Project*. O'Reilly Media, Inc., 2005.
8. X. Ge, Y. Dong, and K. Huang. Shared knowledge construction process in an open-source software development community: an investigation of the gallery community. In *ICLS '06: Proceedings of the 7th international conference on Learning sciences*, pages 189–195. International Society of the Learning Sciences, 2006.
9. I. Herraiz. *A statistical examination of the evolution and properties of libre software*. PhD thesis, Universidad Rey Juan Carlos, 2008. <http://purl.org/net/who/iht/phd>.
10. D. Izquierdo-Cortazar, G. Robles, J. M. González-Barahona, and J.-C. Deprez. Assessing floss communities: An experience report from the qualoss project. In *OSS*, page 364, 2009.
11. D. Izquierdo-Cortazar, G. Robles, F. Ortega, and J. Gonzalez-Barahona. Using software archaeology to measure knowledge loss in software projects due to developer turnover. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS-42)*, Hawaii, USA, January 2009. forthcoming.
12. T. Mens, J. Fernández-Ramil, and S. Degrandart. The evolution of eclipse. In *ICSM*, pages 386–395, 2008.
13. M. Michlmayr, G. Robles, and J. M. Gonzalez-Barahona. Volunteers in large libre software projects: A quantitative analysis over time. In S. K. Sowe, I. G. Stamelos, and I. Samoladas, editors, *Emerging Free and Open Source Software Practices*, pages 1–24. Idea Group Publishing, Hershey, Pennsylvania, USA, 2007.
14. T. Otte, R. Moreton, and H. D. Knoell. Applied quality assurance methods under the open source development model. In *COMPSAC*, pages 1247–1252, 2008.
15. D. L. Parnas. Software aging. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
16. V. T. Rajlich and K. H. Bennett. A staged model for the software life cycle. *Computer*, 33(7):66–71, 2000.
17. G. Robles. Contributor turnover in libre software projects. In *Proceedings of the Second International Conference on Open Source Systems*, 2006.
18. G. Robles, J. M. Gonzalez-Barahona, D. Izquierdo-Cortazar, and I. Herraiz. Tools for the study of the usual data sources found in libre software projects. *International Journal on Open Source Software and Processes*, 1(1), 2008.



# Evolution of Open Source Software Networks

Matthew Van Antwerp

University of Notre Dame [mvanantw@cse.nd.edu](mailto:mvanantw@cse.nd.edu)

**Abstract.** The work presented in this paper is focused on the Open Source Software (OSS) community structure, with particular emphasis on the Concurrent Versions System (CVS) and Subversion (SVN) datasets used to produce the network structure. The theme of the work in this paper is the dynamics of these networks over time. Much research has been done analyzing the network structure at some particular point in time or the network formed by conflating connections made over a specific time span. We present a review of OSS research using CVS/SVN data. We provide an introduction to complex networks as it relates to the open source software community and the numerous studies analyzing networks formed by different software communities, such as SourceForge and GNU Savannah. We also present an initial analysis of the developer and project networks for BerliOS Developer, SourceForge, and GNU Savannah. We also present the results of a study on the importance of social network structure as an indicator of long-term project success. We propose work relating to the evolution of open source software networks as well as the importance of the dynamic network structure.

## 1 Complex Networks

The OSS network is defined as follows. Developers and projects are considered nodes in the graph. If a developer works on a project, there is an edge between the developer and that project. Since developers can only work on projects, the resulting graph will be bipartite, with developers and projects being the two groups having no edges within those groups. This bipartite graph can be easily transformed into a developer network or a project network. From the CVS database, users, groups, and timestamps were extracted. The timestamps are the dates (in unix time) of the oldest and most recent commits to that particular project. With this information, even if two users worked on the same project, a tie was only created between them if they worked on the project at the same time, i.e. their time frame windows overlapped.

## 2 Evolution of Social Networks

In this paper, we present the results of past work as well as an initial study of the social networks of three different forges, BerliOS Developer, GNU Savannah, and SourceForge. Much research has been done on snapshot or conflated views of these networks, especially SourceForge, due to the size of the SourceForge community. The degree distribution, connectedness, centrality, and scale-free nature of SourceForge has been

presented for the network at particular points in time. However, very little research has been done on how the network grows, how connections were made, especially during its infancy, and how these metrics evolve over time. SourceForge is nearly 10 years old and GNU Savannah is older. The CVS archives hosted on these sites go back many decades for some projects. We propose that studying the evolution of these networks can tell us more than any one particular snapshot of the network.

### 3 Importance of Repeated Connections

An aspect of social networks that has been mostly neglected is the occurrence of repeated connections. For example, two developers work on project A and later in time, decide to work together again on project B. While previous studies have concluded that social network structure has no effect on project success, this ignores how developer ties are formed. Work presented in this paper shows that these previously created ties indicate a higher likelihood of success than the average developer tie. Furthermore, these ties also indicate a higher likelihood that the previous collaboration was on a successful project. The incidence of repeat ties (and the original ties that led to those repeats) in SourceForge is 2.65% of all developer-developer ties. The rate is slightly lower on BerliOS, at 2.51%. However, for GNU Savannah, the incidence rate is nearly 10% of all ties. With a small number of projects and a very long history compared to the other two forges, this rate is not completely surprising.

### 4 Summary of Proposed Research

Evolution of OSS developer and project networks has not been extensively and comprehensively studied. While many studies exist examining static or conflated networks, few analyze the change in important network measurements over a period of time. With the data granularity of the CVS/SVN archive, we plan to identify the patterns these network metrics exhibit in the short and long term. The evolution of individual projects will also be studied. Network communities will be identified. The evolution of these network communities over time will also be examined. This work will be done for each of the three forges for which we have data (BerliOS, Savannah, and SourceForge).

### 5 Motivation

In [20], Hinds states in his dissertation that, “it may be appropriate to step back and consider the possibility that open source software communities may be a fundamentally new form of collaborative development.” In his work, he discovered that many of his conjectures from [21], grounded in traditional social network theory, were wrong and some of his results seemed to be counter-intuitive.

Massey studied OSS CVS data and states, “One of the more unique features of open source software development is the continuity of projects over large time scales

and incremental development efforts. For this reason, the open development process provides an interesting environment for investigation of the software development process.” He goes on to tout the research opportunities afforded by such data and the implications such research may have on software productivity and OSS development in general.

In [39], Xu states “the number of sample projects of current OSS research is in the hundreds, which is not big enough to reflect the OSS phenomenon.” The size of the CVS/SVN dataset is large, covering all projects that have used those SCMs and make their code publicly accessible, and the data covers three different forges. Furthermore, in her 2007 dissertation, she states that “Relationships between different OSS projects and developers have not been fully explored.”

We propose completion of the following research:

- Identify communities and their evolutionary trends in the BerliOS, Savannah, and SourceForge developer and project networks.
- Analyze the importance of repeat developer ties in those three forges and the effect they have on project success.
- Identify individual developer evolution trends over their coding lifetimes.
- Analyze the core differences between the three major OSS development networks from static and dynamic perspectives.

The results of such research may tell us how the OSS community can become better connected and improve software productivity.

## 6 OSS Hosting Websites

SourceForge is the world’s largest OSS development website, but BerliOS Developer and GNU Savannah are popular OSS hosting websites as well. BerliOS is a German website with about 1500 users and nearly 3200 projects [2]. GNU Savannah is another hosting site that started when the SourceForge project itself was relicensed as proprietary software [17]. BerliOS and Savannah are smaller than SourceForge, however Savannah in particular is a very tight-knit group of developers with a long history, so the data set is very rich and provides many important research opportunities. Savannah has an enormous number of revisions on extremely mature and widely used OSS projects. CVS and Subversion metadata for all projects on SourceForge, GNU Savannah, and BerliOS was obtained and made available for research [31].

### 6.1 SourceForge Research Data Archive

The SourceForge Research Data Archive is an ongoing project at Notre Dame that makes SourceForge data available to researchers around the world. SourceForge provides SRDA with monthly snapshots of their back-end database. In that database, there is information about users, projects, project development status, target audience, target operating system, programming language, download statistics, web page hits, file

release information, forums, and other statistics. Actual source code is kept in Concurrent Versions System (CVS) or Subversion (SVN) repositories. This data was downloaded and loaded into a database, the details of which are discussed in [31] and [33].

## 7 Existing Research Using CVS/SVN Data

### 7.1 Modification Requests and File Relations

In [16], they analyze numerous statistics related to modification requests (MRs). First they plot the number of MRs per month over the lifespan of the project. They also plot the cumulative percentage of transactions by developers, ordering the data from most active to least active. They plot similar data for number of files per MR. Lastly, they analyze the most often edited files, the top developers by activity, and activity categorized by file extension. In [43], CVS log data were used to identify logical couplings of files. A similar study was done in [42] by Ying, et al for the purpose of identifying relevant related files that may need changing when a MR is made. Van Rysselberghe studied CVS log data to mine what they phrased “frequently applied changes” in [35]. He also visualized CVS log data over time in [36]. Mockus studied version control logs to classify software changes [25]. Gall and others studied CVS log data to identify dependencies in [10].

### 7.2 Social Network Analysis

In [19], they apply social network analysis to CVS data. They analyze the committer network as well as the module network (modules being a subset of a project) by computing degrees, weighted degrees, distance centrality, betweenness centrality, and clustering coefficient (weighted and unweighted). A similar approach is taken in [22] to group contributors into core and peripheral teams. Social network analysis has often been applied to software development, such as in [5], [41], [40], [3], [13], [14], [7], and [30].

In [11], Gao studied the social network of SourceForge and mentioned in future work that a similar study with GNU Savannah would provide more comprehensive conclusions about the OSS social network. Another shortcoming of that thesis was incomplete network data of the SourceForge community. The versioning metadata obtained in [31] provides a complete history of the SourceForge social network as it relates to code committers.

## 8 Previous Work on OSS Networks

Xu in her dissertation [39] analyzed many aspects of the developer and project networks in the SourceForge community. Xu examined the SourceForge developer network over time and determined it to be scale-free [38]. Xu also examined the community structure of the SourceForge developer network in [37] using metrics such as

modularity [27, 26], identifying the largest communities and their populations. Gao examined the diameter, clustering coefficient, centrality, and other metrics of the SourceForge developer network over a timespan of a year and a half [12].

In [19], the authors apply social network analysis to CVS data, graphing network measurements such as degree distribution, clustering coefficient in modules, weighted clustering coefficient, and connection degree of modules for various projects at different time periods in the histories of Apache, Gnome, and KDE. They concluded that both the module network and the developer network exhibit small world behavior.

## 9 OSS Developer and Project Networks

In the following subsections, the developer and project networks of three different forges are discussed.

### 9.1 SourceForge Developer and Project Networks

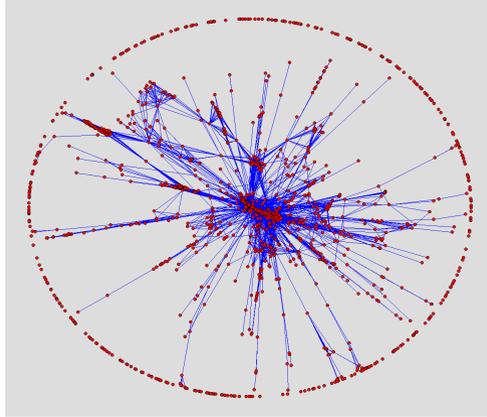
73,829 users have made at least one CVS commit. Of those, 47,946 users are connected to at least one other developer (in other words, they are not the sole developer on all of the projects they work on), which is 64.94%. Of these connected users, the largest connected component contains 19,269 users, which is 40.19%, or 26.10% of all users who have made at least one commit. Due to space constraints, the network visualizations are omitted from this paper. We visualized historical project network snapshots and discovered that a central cluster of projects becomes clear and layers of projects appear surrounding this central cluster.

### 9.2 Savannah Developer Network

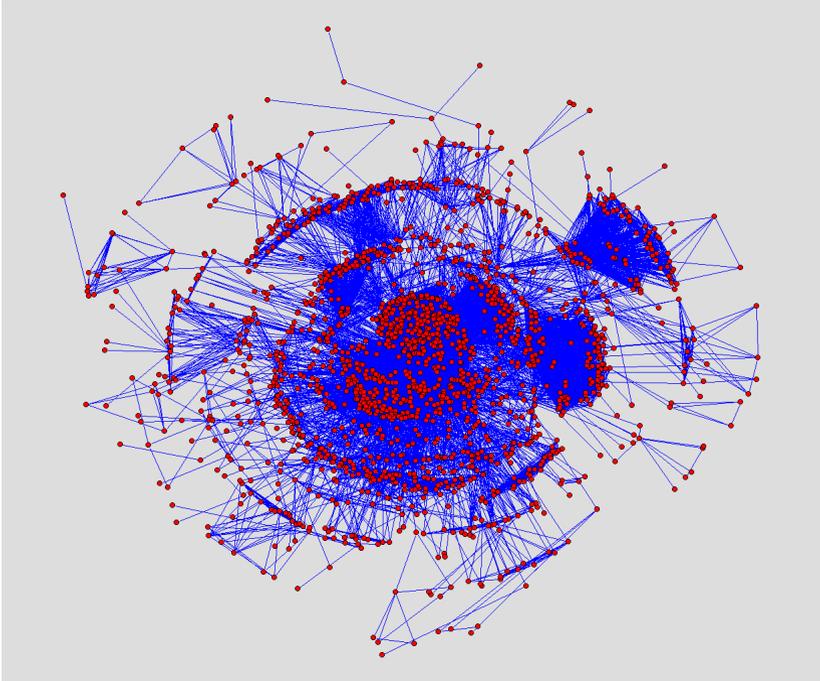
3889 users have made at least one CVS commit. Of those, 3042 users are connected to at least one other developer (they are not the sole developer on all of the projects they work on), which is 78.22%. Of these connected users, the largest connected component contains 1747 users, which is 57.42%, or 44.92% of all users who have made at least one commit. Some network visualizations are provided in figures 1 and 2. In all of these visualizations, developers who are the sole developer on all projects they work on are excluded. They would be singletons in the network were they included. The figure in 1 was produced using the Fruchterman-Reingold algorithm [9]. It is a force-based algorithm where the edges represent attractive forces and there is a repulsive force between vertices. Vertices are moved in an iterative fashion until equilibrium is reached. Visualizations were developed with Pajek [1].

### 9.3 Savannah Project Network

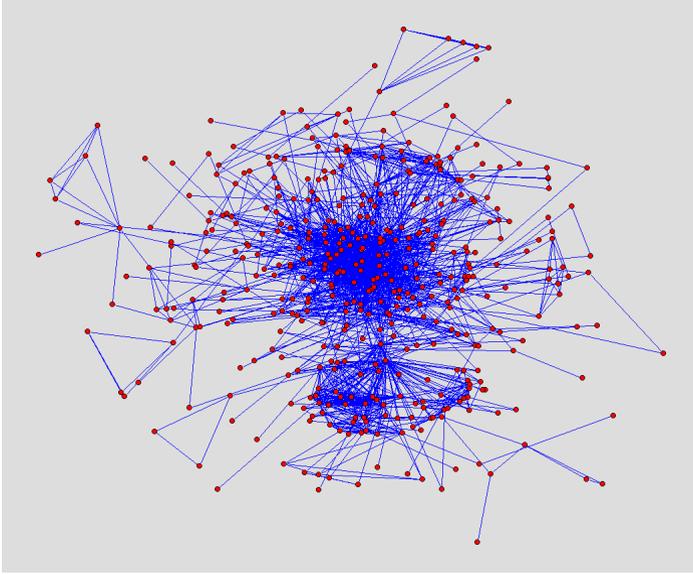
The Savannah project network can be seen in figure 3. The network is well-connected with most projects in one large cluster. This is due to the long life of most Savannah projects, the rarity of new projects hosted at Savannah, and the fact that many developers here work on multiple Savannah projects during their lifetime.



**Fig. 1.** The Savannah developer network. Nodes (red) are developers and edges (blue) are connections between them. The smaller components are displayed on the periphery.



**Fig. 2.** The largest connected component in the Savannah developer network visualized with the Kamada-Kawai algorithm for drawing graphs [24], a force-based algorithm like Fruchterman-Reingold.



**Fig. 3.** The Savannah project network.

#### 9.4 BerliOS Developer and Project Networks

1582 users have made at least one CVS commit. Of those, 1113 users are connected to at least one other developer (they are not the sole developer on all of the projects they work on), which is 70.35%. Of these connected users, the largest connected component contains only 100 users, which is 8.98%, or 6.32% of all users who have made at least one commit. A network visualization is provided in figure 4. Again, developers who are the sole developer on all projects they work on are excluded. The BerliOS project network visualization is omitted. The project network is mostly disconnected. There are however a handful of interesting cliques present in this network.

#### 9.5 Evaluation of the Communities

BerliOS is not a very globally connected developer community. While many developers are connected to someone else, there does not seem to be any sort of small-world effect in this network. SourceForge is a very large community and is better connected than BerliOS. About one quarter of all developers (CVS committers) in SourceForge are in the largest connected component. However, Savannah has nearly half of all developers in the largest connected component, an impressive aspect. A summary of the aforementioned statistics is available in table 1.

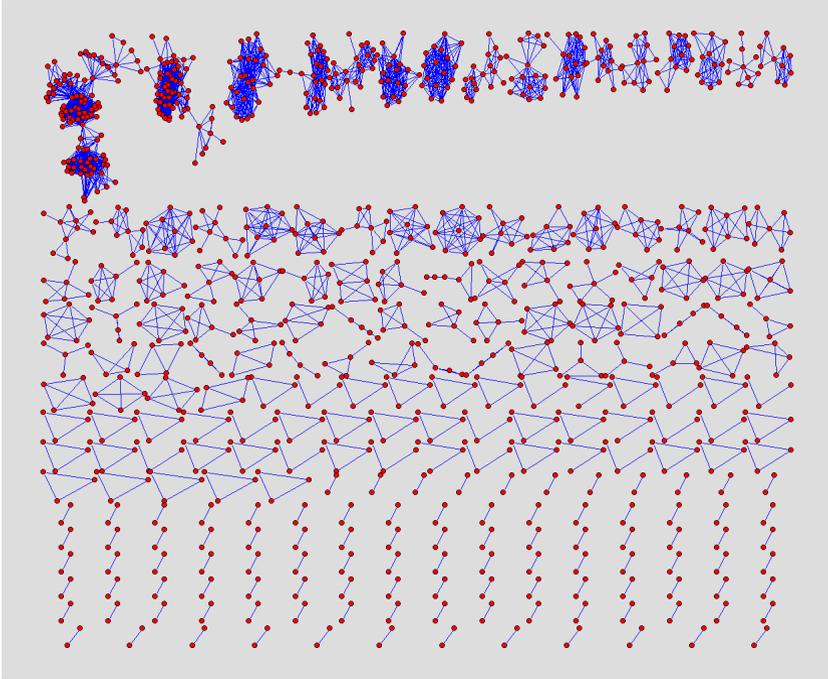


Fig. 4. The BerliOS developer network.

Table 1. Size of largest connected component in the developer networks

Hosting Site	Total Size	Number Connected	% Connected	Largest CC	% of total
SourceForge	73,829	47,946	64.94%	19,269	26.10%
Savannah	3889	3042	78.22%	1747	44.92%
BerliOS	1582	1113	70.35%	100	6.32%

## 10 Proposed Work on OSS Networks

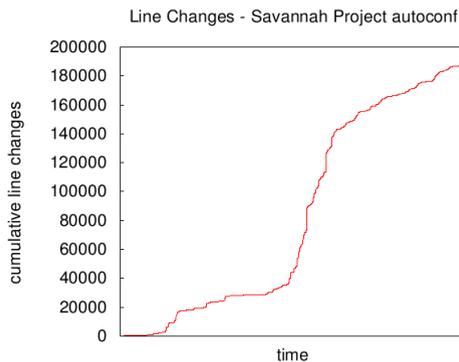
The proposed work for this topic falls under the categories of community detection and identification of development patterns.

### 10.1 Community Detection and Evolution

Communities are defined as groups of nodes in a particular network. Communities of projects as well as developers can be identified in various ways. This has been done to some extent in [39]. Developers can also be classified or clustered by their activity patterns into groups. Using community detection algorithms, such as Clauset’s method for finding community structure in very large networks [6] and Radicchi’s method [29], we intend to identify communities at various points in time for the BerliOS,

Savannah, and SourceForge developer and project networks. Since the CVS/SVN data is as fine-grained as it could possibly be in terms of identifying precisely when a user joins a project (as a developer), there is sufficient opportunity to identify changes in the community structure over time. Communities that are simply projects are likely to be identified, however more interesting patterns are likely to emerge, especially given the hierarchical nature of many community detection algorithms. Furthermore, social network growth in real world social networks tend to follow the three principles put forth in [23]. Those principles are limited for a developer network however. Two developers do not necessarily form a connection simply because they have worked with a mutual developer in the past. Over a long period of time, these connections may be likely, but connections such as those are not likely to occur with such frequency as they do in the real world. We intend to examine the growth of the developer network in light of this and other literature identifying social network evolution trends in the real world.

In [28], Ngamkajornwiwat studied the evolution of OSS developer community from a social network perspective. They determined that the increase in total effort on an OSS project will diminish over time (such a trend can be seen in total line changes in figure 5), the difference in communication participation between core and peripheral developers will decrease over time, and the social network of an OSS project developer network will change in “observable patterns” over time. This was a successful initial study into the KOffice project. I propose to do further evolutionary network analysis of OSS projects. While traditional social network analysis has been applied to OSS network analysis, previous studies suggest that OSS networks differ fundamentally from other networks and may therefore require a new social network analysis paradigm.



**Fig. 5.** The increase in project effort, as measured by total line changes, diminishes over time for the `autoconf` project. The timespan displayed here is from 1992 to 2007.

In conjunction with SRDA data, we intend to determine the evolution of developers over their coding lifetime, in terms of both placement in the social network structure and contribution habits and development roles. Similar work has been from a project-

centric standpoint over a limited timeframe by Christley [4]. Due to the richness and depth of the Savannah dataset in particular, this will identify long-term (up to multi-decade) trends that have heretofore been impossible to display.

## 10.2 Development Patterns

Projects are composed of numerous files. As a project evolves, new files are introduced and development on others may be abandoned. Since CVS/SVN data is granular to the file level, project evolution trends can be identified. Do developers tend to work on the same files? In the short time, it is likely, but long-term trends may be more interesting. Can files be grouped into communities in the same way that developers can? Would hierarchical clustering identify logical file hierarchy? Can file community detection identify poor project structure? An evaluation of this type of linux was done in [18], analyzing the growth trends of different subsystems (e.g. drivers, kernel, lib, etc.). The clear modular structure of the linux project was apparent when the authors analyzed the development trends. Successful projects will likely have modular structure from the start or after refactoring as the source code grows larger and more unwieldy. Crowston and Howison [8] examined projects for such modularity (they use the term decentralization) in a sample of projects, including those hosted at SourceForge and Savannah. They indicate that their results suggest larger projects have a higher tendency to be modularized and that becoming increasingly modular may be a key to growth. However, they did not analyze the evolutionary trends of the data. Examining large, successful projects in their infancy may show when and how projects become more modular, or that they were modular from the beginning.

# 11 The Importance of Repeat Ties in Developer Networks

We presented the results of an initial study into the importance of repeat ties in developer networks at HICSS-43 [34]. We present the essential aspects of that research below and outline future work that builds off of our previous research.

Previous work done by Hinds [21] suggested social network structure was instrumental towards the success of an OSS project, as measured by activity and output. The follow-up paper by Hinds [20] discovered that his hypotheses, based on social network theory and previous research on the importance of subgroup connectedness, were vastly different than the results of his study of over 100 successful OSS projects. He concluded that the social network structure had no significant effect on project success. We outline how his approach disregarded potentially important factors and through a new study evaluate the role of the OSS developer network as it pertains to long-term project popularity. In contrast with Hinds, this study shows that previously existing developer-developer ties are an indicator of past and future project popularity.

## 11.1 Method

Using data from the SourceForge Research Data Archive [15, 32] and the new dataset of concurrent versions system (CVS) and Subversion (SVN) metadata described in

[31], we analyzed how previous network ties affect future developer communities. First, the percentage of ties that existed on previous projects was identified. Then, we determined which projects have long-term popularity and identified whether or not previously-existing developer community ties increase the odds of producing such projects. To measure long-term popularity, we used the SourceForge activity percentile. Activity percentile will be high if the project is popular since it measures recent traffic, development, and communication. If the project is not popular in the long run, this will fall down towards zero as development and downloads diminish. Activity percentile was taken from a month after the last developer-developer tie was formed from the CVS dataset.

## 11.2 Hypotheses

We hypothesize that that popular projects are more likely to contain ties that existed on previous projects than the average project. Furthermore, ties made on previous projects are more likely to lead to popular projects than projects with no previous developer ties.

## 11.3 Results

Data extracted from the developer-developer network and activity percentiles for the projects on which those ties were formed supports these hypotheses.

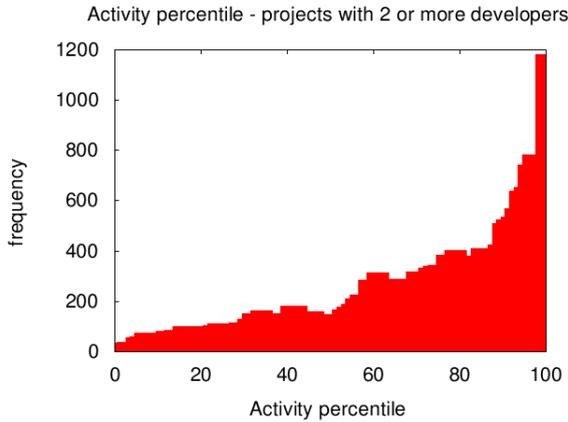
**Developer Network Statistics** Figure 6 contains the distribution for projects that have at least two developers. This is a baseline for comparison since developer-developer links are only possible on projects that have at least two developers. The average percentile for these projects is 70.68. Figure 7 contains the distribution for projects that contain either repeat links or links that were repeated on a project at a later time, where the average percentile is 76.81. As hypothesized, these projects tend to have higher activity percentile than the typical project that has at least two developers.

## 11.4 Conclusions

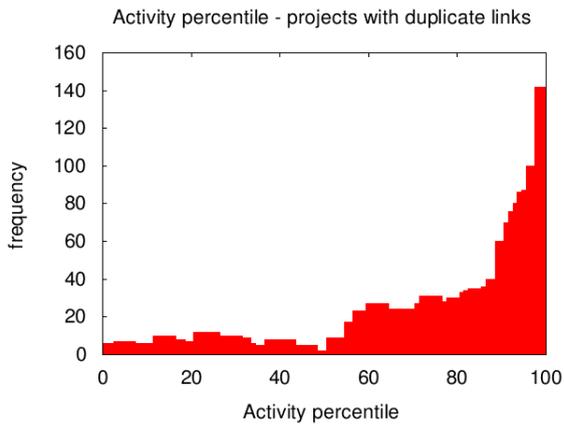
Previous ties are generally an indicator of past success and usually lead to future success. When a repeated developer-developer link appears, it tells us something about both projects linking the two developers. Generally, the previous project was successful on some level. The two developers likely worked together well. Since they decided to work together again, this means they probably expect the experience to be similar to previous experiences and again form a successful collaboration.

## 11.5 Future Developer Ties Work

Developer-developer connections are binary for the purposes of this study. However, they can be weighted with many possible values. The amount of overlap in their respective time-windows on the project at hand, the number of commits made by each



**Fig. 6.** Distribution of activity percentile for projects with at least 2 developers. March 2008 data used. Average is 70.68.



**Fig. 7.** Distribution of activity percentile for projects with repeat links or links that would later be repeated. March 2008 data used. Average is 76.81.

user, even more fine-grained analysis such as which files they worked on, or combinations of those aspects. Two developers who frequently commit changes to the main project source file indicates a stronger connection than two developers who have never committed changes to the same file on a project but instead work on disjoint parts of the same project. All of this information can be extracted from the CVS database archive. We put forth that long-term project popularity, as measured by the SourceForge activity percentile, is a rough success metric.

## References

1. Vladimir Batagelj and Andrej Mrvar. Pajek - program for large network analysis. *Connections*, 21:47–57, 1998.
2. BerliOS Developer. <http://developer.berlios.de>.
3. Scott Christley and Greg Madey. An algorithm for temporal analysis of social positions. In *NAACSOS2005*, Notre Dame, IN, June 2005.
4. Scott Christley and Greg Madey. Analysis of activity in the open source software development community. In *The 40th Hawaii International Conference on System Sciences*, Hawaii, January 2007.
5. Scott Christley and Greg Madey. Global and temporal analysis of social positions at sourceforge.net. In *The Third International Conference on Open Source Systems (OSS 2007)*, Limerick, Ireland, June 2007.
6. Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70(6):066111, Dec 2004.
7. Kevin Crowston and James Howison. The social structure of free and open source software development. In *First Monday*, 2005.
8. Kevin Crowston and James Howison. Hierarchy and centralization in free and open source software team communications. In *Knowledge Technology & Policy*, volume 18, pages 65–85, 2006.
9. Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, 1991.
10. Harald Gall, Mehdi Jazayeri, and Jacek Krajewski. Cvs release history data for detecting logical couplings. In *IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution*, page 13, Washington, DC, USA, 2003. IEEE Computer Society.
11. Yongqin Gao. Topology and evolution of the open source software community. Master's thesis, University of Notre Dame, 2003.
12. Yongqin Gao. *Computational Discovery in Evolving Complex Networks*. PhD thesis, University of Notre Dame, 2007.
13. Yongqin Gao and Greg Madey. Project development analysis of the oss community using ts mining. In *NAACSOS2005*, Notre Dame, IN, June 2005.
14. Yongqin Gao and Greg Madey. Network analysis of the sourceforge.net community. In *The Third International Conference on Open Source Systems (OSS 2007)*, Limerick, Ireland, June 2007.
15. Yongqin Gao, Matthew Van Antwerp, Scott Christley, and Greg Madey. A research colloboratory for open source software research. In *FLOSS '07: Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development*, page 4, Washington, DC, USA, 2007. IEEE Computer Society.
16. Daniel German and Audris Mockus. Automating the measurement of open source projects. In *Proceedings of ICSE 03 Workshop on Open Source Software Engineering*, Portland, Oregon, 2003.
17. GNU Savannah. <http://savannah.gnu.org>.
18. Michael W. Godfrey and Qiang Tu. Evolution in open source software: A case study. In *In Proceedings of the International Conference on Software Maintenance*, pages 131–142, 2000.
19. Luis Lopez-Fernandez Gregorio. Applying social network analysis to the information in cvs repositories. In *Proceedings of the First International Workshop on Mining Software Repositories (MSR 2004)*, Edinburgh, UK, 2004.

20. David Hinds. *Social Network Structure as a Critical Success Condition for Open Source Software Project Communities*. PhD thesis, Florida International University, 2008.
21. David Hinds and Ronald M. Lee. Social network structure as a critical success condition for virtual communities. In *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 323, Washington, DC, USA, 2008. IEEE Computer Society.
22. Shih-Kun Huang and Kang min Liu. Mining version histories to verify the learning process of legitimate peripheral participants. In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, New York, NY, USA, 2005. ACM.
23. Emily M. Jin, Michelle Girvan, and M. E. J. Newman. Structure of growing social networks. *Physical Review E*, 64(4):046132+, 2001.
24. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, April 1989.
25. Audris Mockus and Lawrence G. Votta. Identifying reasons for software changes using historic databases. In *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, page 120, Washington, DC, USA, 2000. IEEE Computer Society.
26. M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69:066133, 2004.
27. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004.
28. Kawin Ngamkajornwiwat, Dongsong Zhang, A. Gunes Koru, Lina Zhou, and Robert Nolker. An exploratory study on the evolution of oss developer communities. In *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 305, Washington, DC, USA, 2008. IEEE Computer Society.
29. Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *PROC.NATL.ACAD.SCI.USA*, 101:2658, 2004.
30. Ilkka Tuomi. Internet, innovation, and open source: Actors in the network. In *First Monday*, 2000.
31. Matthew Van Antwerp. Studying open source versioning metadata. Master's thesis, University of Notre Dame, Notre Dame, IN, April 2009.
32. Matthew Van Antwerp and Greg Madey. Advances in the sourceforge research data archive. In *Workshop on Public Data about Software Development (WoPDaSD) at The 4th International Conference on Open Source Systems*, Milan, Italy, 2008.
33. Matthew Van Antwerp and Greg Madey. Warehousing, mining, and querying open source versioning metadata, 2008.
34. Matthew Van Antwerp and Greg Madey. The importance of social network structure in the open source software developer community. In *The 43rd Hawaii International Conference on System Sciences (HICSS-43)*, Hawaii, January 2010.
35. Filip Van Rysselberghe and Serge Demeyer. Mining version control systems for facts (frequently applied changes). In *Proceedings MSR'04 (International Workshop on Mining Software Repositories)*, pages 48–52. IEE (Institution of Electrical Engineers), 2004.
36. Filip Van Rysselberghe and Serge Demeyer. Studying software evolution information by visualizing the change history. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 328–337, Washington, DC, USA, 2004. IEEE Computer Society.
37. J. Xu, S. Christley, and G. Madey. The open source software community structure. In *NAACSOS2005*, Notre Dame, IN, June 2005.
38. J. Xu and G. Madey. Exploration of the open source software community. In *NAACSOS 2004*, Pittsburgh, PA, June 2004.

39. Jin Xu. *Mining and Modeling the Open Source Software Community*. PhD thesis, University of Notre Dame, 2007.
40. Jin Xu, Scott Christley, and Greg Madey. *Application of Social Network Analysis to the Study of Open Source Software*. Elsevier Press, 2006.
41. Jin Xu, Yongqin Gao, Scott Christley, and Greg Madey. A topological analysis of the open source software development community. In *The 38th Hawaii International Conference on System Sciences (HICSS-38)*, Hawaii, January 2005.
42. Annie T. T. Ying, Raymond Ng, and Mark C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Trans. Softw. Eng.*, 30(9):574–586, 2004. Member-Gail C. Murphy.
43. Thomas Zimmermann, Stephan Diehl, and Andreas Zeller. How history justifies system architecture (or not). In *In IEEE International Workshop on Principles of Software Evolution (IWPSE03)*, pages 73–83. IEEE Computer Society, 2003.



# Behind Linus's Law: Investigating Creative Peer Review Processes in Open Source

Jing Wang

College of Information Sciences and Technology  
The Pennsylvania State University  
University Park, PA US  
jzw143@ist.psu.edu

**Abstract.** Linus's law highlights *extensive peer review* as the advantage of open source software development. Despite the success of open source and the importance of peer review, research has not examined peer review processes or their relationship with creativity in open source communities. This project fills the gap by investigating how peer review is conducted and how peer review supports creative collaboration in open source communities. It will use comparative case studies with content analysis and interviews to develop conceptual and empirical analyses on open source peer review practices.

## 1 Introduction

“Given enough eyeballs, all bugs are shallow”, Linus's law [1], is a popular belief among open source software (OSS) researchers and practitioners. It highlights *extensive peer review* as an advantage of open source software development. OSS involves geographically dispersed experts collaborating over the Internet to produce software that can be used for free. Its success represents a new and appealing model of creative collaborative knowledge work. Peer review involves critical evaluation on the peers' products. Feedback from reviews may lead to product improvement.

Despite the importance of peer review, current OSS research lacks investigations on review processes: (1) Linus's law has not been questioned, or examined in depth; (2) peer review's potential support for creativity of open source communities has been overlooked. First, very few OSS studies analyzed peer review in particular. None of the studies explains how the review processes achieve effectiveness, or why they are superior to those in closed software development projects. Linus's law may have overestimated the scale of participation in OSS review. Second, OSS research has not discussed the relationship between peer review and creativity. Peer review is important for creativity [2]. Feedback from peers enables developers to critically rethink their design or implementation, and then improve it or even create a new one. Recognition from the community will also motivate developers to contribute more

creative products. This evaluation and reward system works similarly as the one in scientific communities [3]. Creativity emerges from such systems [4].

The objective of this project is to characterize peer review processes and to codify their patterns that support creativity in open source communities. My investigation will involve reviews not only on code, but also on other work products during development processes, such as user interface designs and prototypes, requirements specifications and project plans.

This project will adopt a comparative case studies approach. Two well-established and creative OSS organizations, Mozilla and Python, are selected as the cases. Most of peer review activities are externalized and recorded in communities' online public repositories, for instance, mailing lists and issue tracking systems. Therefore, this project will apply the content analysis method to study the cases both qualitatively and quantitatively. Interviews will also be used for triangulation.

This project will contribute to several research fields. First, it examines Linus's law and provides rich evidence of how open source communities conduct their peer review. Second, it will benefit other peer review systems. A few research and practical attempts aim at improving peer review processes in software development and scientific research. Most of them failed. Understanding of open source peer review may shed light on designing peer review practices in these contexts. Third, it adds in-depth studies in creativity research, which calls for such studies [5]. The relationship between peer review and creativity was only discussed conceptually. This project will elaborate how reviews affect creativity.

The proposal is organized as follows. Section 2 describes characteristics of open source to establish an understanding of the study context. Section 3 reviews the previous work on software peer review. Section 4 presents a theoretical framework adapted from creativity research. This framework will guide the investigation in this project. Section 5 explains the research design and methodology for this project.

## **2 Open Source**

Open source is a multi-facet concept. From software development perspective, it can be characterized by its motivated developers, high modularity, informal and implicit requirements and design, extensive peer review, and rapid iteration and release [6-11]. From organization perspective, it can be featured by its hybrid structure, diverse and dynamic expertise, encouraging reward systems, and open flow of information exchange [3, 11-14]. Given the limit of space, this section does not discuss the details of these characteristics.

## **3 Peer Review**

Peer review is a process that one's performance or the quality of work products is evaluated by other colleagues. It is used extensively in various fields, such as

scientific research, software development, and healthcare. Its purpose is to detect flaws in one's work and discover potential for further improvement. Current research on scientific peer review focuses on reducing review bias.

Software peer review is primarily used to discover defects (or bugs) as early as possible during development processes, suggest improvements and even help developers create better products [15]. Other than revealing that an execution outcome fails to meet expectation, peer review can also evaluate other dimensions of quality, such as understandability, maintainability and conciseness. Furthermore, it can facilitate communication and knowledge sharing among project members.

Code is not the only object that peer review can assess. Review candidates can be any artifacts created during the development process, such as requirements specifications, use cases, project plans, user interface design and prototypes, and program documentation [16].

### **3.1 Peer Review in Traditional Software Development**

Peer review can take a variety of forms with different degrees of formality and flexibility [17]. *Inspections* are the most systematic and rigorous review, introduced by Fagan's salient work [18]. An inspection has a clearly specified process, including planning, overview, preparation, meeting, rework, follow-up, and causal analysis. In inspections, five types of roles are often engaged: the inspection leader/moderator, the recorder, readers, the author, and inspectors [16].

A large number of quantitative metrics can be used for peer review. Boehm et al. [19] defined a software quality characteristic tree to guide software quality assessment, including metrics such as device-independence, robustness, completeness, conciseness, and consistency.

Several issues on peer review processes are being discussed. One is the effective team size of peer review [20-21]. Another is about the necessity of a meeting. Holding a meeting may enable collaboration on discovering new defects and synthesize their findings. However, some studies indicate that the effectiveness of meeting-based reviews is similar to that of asynchronous reviews without a meeting [22], or even worse [15, 23].

Techniques for detecting defects include checklists, scenarios, rule sets, and ad hoc reading [15, 24]. Researchers also have developed computer-supported tools to reduce the labor cost and enhance the effectiveness of peer review [25-26].

### **3.2 Peer Review in Open Source Software Development**

Although extensive peer review is considered a salient advantage of OSSD, very little research has investigated review activities. It only reported some general statistics [27-29], described documented policies [30], or developed extraction methods and quantitative metrics to analyze online content of peer review [29, 31-32]. Most of that research focuses on code review [33], which is often triggered by a

“check-in” action or a bug report. The other part discusses the reviews on design and usability issues [34].

Some large OSS projects have a statement of their review criteria, although they vary with the projects’ philosophies. Other than traditional software quality metrics, evaluation may also be concerned with elegance, developers with a high level of reputation, distributions, and tools indicating the level of activities [35].

Generally, OSS code review consists of two kinds of procedures . One is Review-Then-Commit, which requires a submitted patch being reviewed by other developers before it is applied to the project. It is often used for significant change of the software system. The other is Commit-Then-Review, which is usually applied to relatively trivial and simple modifications to the system.

Roles involved in OSS peer review mostly include a bug reporter, a contributor, a tester, a reviewer, and a committer [29]. Some projects have an explicit assignment of reviewers, usually including at least one core member. Some other projects only adopt an autonomous way to share the responsibility.

OSS peer review largely relies on computer-supported tools, including bug tracking systems/issue trackers, source configuration management systems, mailing lists, discussion forums, instant messaging, and so forth.

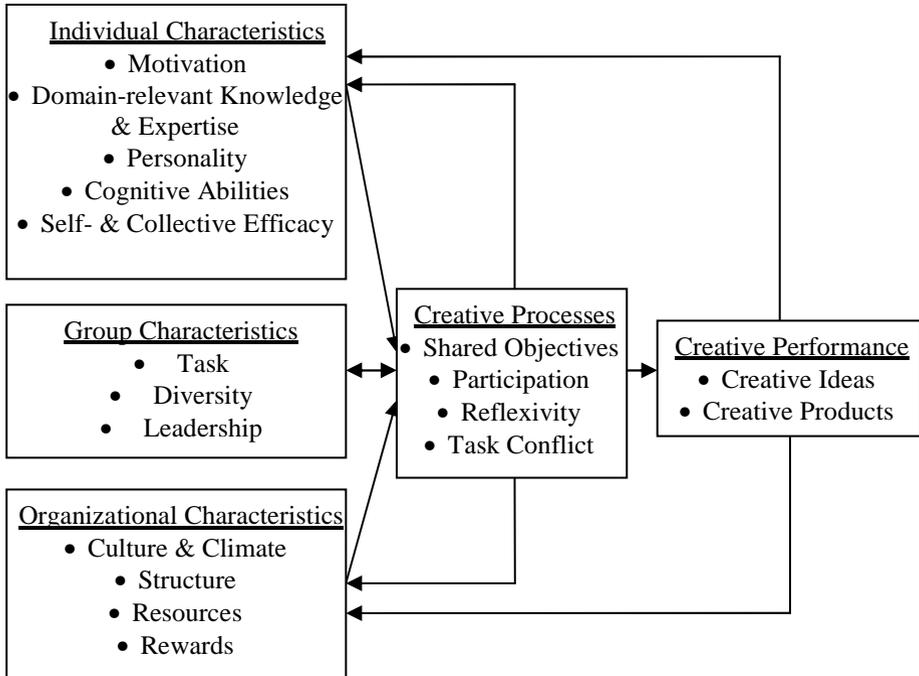
## 4 Theoretical Lens: Creativity in Collaborative Work

Creativity, in this project, emphasizes the “little C” [36], which refers to the everyday problem solving and the adaptability to changes . This project is not intended to verify any organizations as creative, but rather to analyze practices of the ones that are well-recognized as creative organizations. These practices should contribute to the organizational creativity to some extent.

To facilitate the analysis, this project adapts Amabile’s model of creativity and innovation in organizations [37], Woodman’s interactionist model of organizational creativity [38] and West’s model of work group innovation [39] into a framework. These models comprehensively encompass the key factors affecting organizational creativity, which are well supported by creativity research. Figure 1 illustrates the framework.

The framework integrates multiple variables related to creativity at individual, group and organization levels. Note that these variables are not exhaustive, nor are they completely distinct from each other. Furthermore, rather than conceptualizing these components in a linear input-process-output model as Woodman and West did, I situate them in an evolving system: the outcome of creative processes is not only externalized into creative products, but also reshapes characteristics of individuals, groups and organizations.

This project will focus on the creative processes. Given the limit space here, I only discuss factors related to the processes and list factors related to other components in Figure 1.



**Fig. 1.** Adapted Theoretical Framework for Analyzing Creativity in Open Source

#### 4.1 Shared Objectives

Clear objectives will facilitate innovation if members are committed to the goals. This strong goal commitment is necessary to keep members' persistence for implementation even when they are facing resistance from the environment [39].

#### 4.2 Participation

Participation in decision making facilitates information flow with the organization [38]. It also strengthens members' sense of power and control within the organization, which in turn will enhance their intrinsic motivation [40]. Pooling unique information can lead to cross-fertilization of perspectives that can spawn creativity and innovation [41].

### 4.3 Reflexivity

Group reflexivity include reflection, planning and action [42]. It is the extent to which members collectively reflect on the group's objectives, strategies, and processes as well as their wider organizations and environments, and adapt them accordingly.

### 4.4 Task Conflict

Constructive conflicts in collaborative work can improve the quality of decision making and creativity [41, 43]. They can overcome the obstacles inhibiting creativity including groupthink [44], satisficing [45], and organizational inertia.

Task conflicts, as one type of conflicts, arise from concerns with the quality of task performance, and appraisal and critical thinking of group processes and performance. Its effects on creativity may be altered by moderators, such as the team climate, the task types to be performed, norms with regard to conflict, and within-team trust [46].

## 5 Research Design and Methodology

### 5.1 Research Objective

My objective is to codify the theoretical factors (as discussed in Section 4) in real settings, the work practices and techniques of peer review in the two open source communities (Mozilla and Python), to describe how peer review processes support creativity and innovation for open source, and to inform of the design of computing infrastructure for creative peer review and the design of peer review processes in other organizations. Additionally, this project will further refine our previous *Creative Collective Efficacy* scale, which is intended to assist assessing creative collaboration [47].

#### 5.1.1 Overall Question

RQ: How do open source communities conduct peer review? What peer review practices support creativity of open source communities and how? How do computer-supported tools mediate these processes?

#### 5.1.2 Decomposed Questions

RQ1: How do the shared community values and evaluation criteria affect open source creative collaboration?

RQ1a: What are the community values and evaluation criteria?

RQ1b: How are the community values and evaluation criteria shared in peer review activities?

RQ1c: How do the answers to RQ1a and RQ1b affect the creativity of the author, reviewers, and the community?

RQ2: How does participation in peer review practices affect open source creative collaboration?

RQ2a: Who participate in open source peer review?

RQ2b: What roles do the participants play?

RQ2c: What extent is the participation in peer review?

RQ2d: How do the answers to RQ2a, RQ2b and RQ2c affect the creativity of the author, reviewers, and the community?

RQ3: How does reflexivity affect open source creative collaboration?

RQ3a: What is the extent to which the review team collectively reflects on their objectives, strategies, processes and environment? How are its members aware of each other's activities?

RQ3b: How do reflections from one review team affect others in the community? How are other community members aware of these reflections?

RQ3c: How do the answers to RQ3a, RQ3b and RQ3c affect the creativity of the author, reviewers, and the community?

RQ4: How do conflicts affect open source creative collaboration? How do they interact with the other factors (e.g., shared objectives and participation)?

RQ4a: What is the extent of the weakness detected in the product being reviewed?

RQ4b: How do the conflicts arise and develop?

RQ4c: How does the review team manage the conflicts? What are the outcomes?

RQ4d: How do the answers to RQ4a, RQ4b and RQ4c interact with the other factors (e.g., shared objectives and participation)?

RQ4e: How do the answers to RQ4a, RQ4b and RQ4c affect the creativity of the author, reviewers, and the community?

## 5.2 Research Design and Methodology

### 5.2.1 Operationalizing the Theoretical Framework into Variables

To answer those aforementioned research questions, I operationalize the theoretical framework into mid-level variables that are specific to open source peer review. Table 1-4 summarize them. Note that these variables are not exhaustive. They are deductive, but this project is still open to inductive variables that may emerge during the project.

**Table 1.** Variables for Individual Characteristics

<b>Factors</b>	<b>Operationalized Variables</b>
Motivation	Intrinsic motivation Informational or enabling motivation Controlling extrinsic motivation
Domain-relevant Knowledge and Expertise	Reputation in the field Length of participation Experience in software design/development
Personality Cognitive Abilities Efficacy	Refer to Adjective Check List [48] Infeasible to collect in this project Creative self-efficacy Creative collective efficacy

**Table 2.** Variables for Group Characteristics

<b>Factors</b>	<b>Operationalized Variables</b>
Task	Software system type (e.g., operating systems, script languages, web servers, IDEs, and web browsers) Product reviewed type (e.g., code, design proposals, and plans)
Diversity Leadership	Skills (e.g., network infrastructure, security, and usability) Single leadership or shared leadership Selection of leaders (e.g., how one becomes a leader) Extent of participation in work Power for making decision

**Table 3.** Variables for Organizational Characteristics

<b>Factors</b>	<b>Operationalized Variables</b>
Culture and Climate Structure Resources Rewards	Refer to KEYS [49]

**Table 4.** Variables for Creative Processes

<b>Factors</b>	<b>Operationalized Variables</b>
Shared Objectives	Community values Expected review outcomes Evaluation criteria
Participation	Extent of Participation Roles Commitment (e.g., assigned and voluntary)
Reflexivity	Reflection Plan Action
Task Conflict	Criticism on the product Disagreement to a review comment Disagreement to a non-review comment

This project will adopt a *comparative case studies* approach. Case studies have the advantage of exploring “how” and “why” questions in complex social contexts over which control is usually impossible [50]. Instead of studying a single case, this project will analyze two cases to highlight theoretical consistencies. The criteria for selecting these two cases are based on Yin’s [50] concept of the logical of theoretical replication. Table 1 presents comparison between the two organizations. It is based on my preliminary analysis on them.

This project will employ multiple methods, mainly including *content analysis* and *interview* for triangulation. *Content analysis* will be used to analyze the creative processes, identify group creative characteristics and to determine the extent of several variables, such as participation and conflict. Content in this project is not limited to discourses but also includes activity logs and software code. Data for content analysis in this project will be gathered from source configuration management systems, issue trackers, bug tracking systems, mailing lists, and discussion forums. Quantitative measurements and extraction methods for content analysis can be adapted from several previous studies [29, 31-32]. *Interviews* will be used to clarify the ambiguity and suspects arising from content analysis. They will describe how each role involved in the review activities perceive and interpret the process. Semi-structured interviews will be conducted in this project with some standard questionnaires to capture individual characteristics of interviewees, such as personalities.

**Table 5.** Comparison between Mozilla and Python (up to date March 7, 2010)

<b>Characteristics</b>	<b>Mozilla</b>	<b>Python</b>
Age	11 years	18 years
Stage of the Main Project	Firefox 3.6	Python 2.6 and Python 3.1
Values	Openness Innovation Opportunity	Usability Easy to learn Simple
Products	Web browser Web-related applications	Computer programming language
Roles	Module owners, super-reviewers, release drivers, Bugzilla component owners, and benevolent dictators	Module owners, core developers, sponsors, and benevolent dictators
Structure	Mozilla Foundation, Mozilla Corporation and Mozilla Messaging	Python Software Foundation
Peer Review	Review and super-review	Patch review and commit review
Tools for General Communication	Mailing lists Blogs Wiki Web Portal Internet Relay Chat Discussion forums RSS Feeds	Mailing lists Blogs Wiki Web Portal Internet Relay Chat Discussion forums Newsgroups
Tools for Software Development	Version control system (Mercurial) Bug tracking system (Bugzilla) CVS archival searching (Bonsai) CVS status tool (Tinderbox) Etc.	Version control system (SVN) Issue tracker

**References**

1. Raymond, E.: The Cathedral and the Bazaar. Knowledge, Technology, and Policy 12 (1999) 23-49
2. Bergquist, M., Ljungberg, J., Lundh-Snis, U.: Practising peer review in organizations: a qualifier for knowledge dissemination and legitimization. Journal of Information Technology 16 (2001) 99-112
3. Bonaccorsi, A., Rossi, C.: Why Open Source software can succeed. Research Policy 32 (2003) 1243-1258

4. Csikszentmihalyi, M.: Implications of a Systems Perspective for the Study of Creativity. In: Sternberg, R.J. (ed.): Handbook of Creativity. Cambridge University Press, New York (1999) 313-335
5. Shneiderman, B.: Creativity support tools: accelerating discovery and innovation. *Communications of the ACM* 50 (2007) 20-32
6. O'Reilly, T.: Lessons from open-source software development. *Communications of the ACM* 42 (1999) 32-37
7. Raymond, E.S.: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly (2001)
8. Torvalds, L.: The Linux edge. *Communications of the ACM* 42 (1999) 38-39
9. Scacchi, W.: Understanding the requirements for developing open source software systems. *IEEE Proceedings-Software* 149 (2002) 24-39
10. Koch, S., Schneider, G.: Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal* 12 (2002) 27-42
11. Mockus, A., Fielding, R.T., Herbsleb, J.: A case study of open source software development: the Apache server. *Proceedings of International Conference on Software Engineering* (2000)
12. Lerner, J., Tirole, J.: The Simple Economics of Open Source. (2000)
13. Koch, S., Schneider, G.: Results from Software Engineering Research into Open Source Development Projects Using Public Data. (2000)
14. Bergquist, M., Ljungberg, J.: The power of gifts: organizing social relationships in open source communities. *Information Systems Journal* 11 (2001) 305-320
15. Wieggers, K.: Peer reviews in software: A practical guide. Addison-Wesley (2001)
16. The Institute of Electrical and Electronics Engineers, I.: IEEE Guide Standard for Software Reviews. IEEE Std 1028-1997., New York (1999)
17. Iisakka, J., Tervonen, I.: Painless improvements to the review process. *Software Quality Journal* 7 (1998) 11-20
18. Fagan, M.: Design and code inspections to reduce errors in program development. *IBM Journal of Research and Development* 15 (1976) 182-211
19. Boehm, B., Brown, J., Lipow, M.: Quantitative evaluation of software quality. IEEE Computer Society Press Los Alamitos, CA, USA (1976) 592-605
20. Grady, R.: Practical software metrics for project management and process improvement. Prentice-Hall, Inc. Upper Saddle River, NJ, USA (1992)
21. Gilb, T.: Principles of software engineering management. Addison-Wesley Longman Publishing Co., Inc. , Boston, MA, USA (1988)
22. Johnson, P., Tjahjono, D.: Assessing software review meetings: A controlled experimental study using CSRS. (1997) 118-127
23. Nunamakar, J., Dennis, A., Valacich, J., Vogel, D., George, J.: Electronic meeting systems to support group work. *Communications of the ACM* 34 (1991) 40-61
24. Ciolkowski, M., Laitenberger, O., Biffel, S.: Software reviews: The state of the practice. *IEEE Software* (2003) 46-51
25. Brothers, L., Sembugamoorthy, V., Muller, M.: ICICLE: groupware for code inspection. ACM New York, NY, USA (1990) 169-181
26. Johnson, P., Tjahjono, D.: Improving software quality through computer supported collaborative review. The 3rd European Conference on Computer Supported Cooperative Work. Kluwer Academic Publishers, Dordrecht, Netherlands (1993) 61-76
27. Asundi, J., Jayant, R.: Patch Review Processes in Open Source Software Development Communities: A Comparative Case Study. HICSS' 07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences (2007) 166c

28. Nuroolahzade, M., Nasehi, S., Khandkar, S., Rawal, S.: The role of patch review in software evolution: an analysis of the mozilla firefox. IWPSE-Evol' 09: Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops. ACM (2009) 9-18
29. Rigby, P., German, D., Storey, M.: Open source software peer review practices: a case study of the apache server. ICSE'08: Proceedings of the 30th international conference on Software Engineering. ACM New York, NY, USA (2008) 541-550
30. Johnson, J.P.: Collaboration, peer review and open source software. Information Economics and Policy 18 (2006) 477-497
31. Barcellini, F., Détienne, F., Burkhardt, J.M., Sack, W.: A study of online discussions in an Open-Source Software Community. Proceedings of the Communities and Technologies 2005 conference, Springer Verlag (2005)
32. Mockus, A., T Fielding, R.O.Y., D Herbsleb, J.: Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology 11 (2002) 309-346
33. Crowston, K., Scozzi, B.: Coordination practices within FLOSS development teams: The bug fixing process. Computer Supported Activity Coordination 4 (2004) 21-30
34. Twidale, M.B., Nichols, D.M.: Exploring Usability Discussions in Open Source Development. Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05), Vol. 28 (2005)
35. de Joode, R.: Innovation in open source communities through processes of variation and selection. Knowledge, Technology & Policy 16 (2004) 30-45
36. Gardner, H.: Creating minds. Basic Books New York (1993)
37. Amabile, T.M.: A model of creativity and innovation in organizations. Research in Organizational Behavior 10 (1988) 123-167
38. Woodman, R.W., Sawyer, J.E., Griffin, R.W.: Toward a theory of organizational creativity. Academy of Management Review 18 (1993) 293-321
39. West, M.A.: Innovation implementation in work teams. In: Paulus, P.B., Nijstad, B.A. (eds.): Group Creativity: Innovation Through Collaboration. Oxford University Press, New York (2003) 245-276
40. Amabile, T.M.: How to kill creativity. Harvard business review 76 (1998) 76-87
41. Mumford, M.D., Gustafson, S.B.: Creativity syndrome: Integration, application, and innovation. Psychological bulletin. 103 (1988) 27-43
42. West, M.A.: Reflexivity and work group effectiveness: A conceptual integration. Handbook of work group psychology (1996) 555-579
43. Johnson, D., Johnson, R., Tjosvold, D.: Constructive controversy. The handbook of conflict resolution: Theory and practice (2000) 65
44. Janis, I.L.: Groupthink. Houghton Mifflin Boston, Boston (1982)
45. Simon, H.: Models of man. Wiley, New York (1957)
46. Simons, T., Peterson, R.: Task conflict and relationship conflict in top management teams: The pivotal role of intragroup trust. Journal of Applied Psychology 85 (2000) 102-111
47. Wang, J., Farooq, U., Carroll, J.M.: Creative Collective Efficacy in Scientific Communities. COOP'10: 9th International Conference on the Design of Cooperative Systems Springer, Aix en Provence, France (2010)
48. Gough, H.G.: A creative personality scale for the Adjective Check List. Journal of Personality and Social Psychology 37 (1979) 1398-1405
49. Amabile, T.M., Conti, R., Coon, H., Lazenby, J., Herron, M.: Assessing the Work Environment for Creativity. The Academy of Management Journal 39 (1996) 1154-1184
50. Yin, R.: Case study research: Design and methods. Sage Publications, Inc (2008)

# The Role of Requirements in Open Source IT Innovation

Daniel Curto Millet

London School of Economics and Political Science  
Houghton Street, London WC2A 2AE, UK  
d.a.curto-millet@lse.ac.uk

**Abstract.** The requirements process is a collection of activities in software engineering which purpose it is to define the functionality and properties of a future system-to-be, taking into account the environment and the problem domain. However simple as it may sound, it is a difficult undertaking, and it would seem that OS requirements are handled in a particular manner. Further, requirements can be seen as being intimately linked to IT innovation as representing attempts to create a certain vision of a future IT system. If open source organisations are as different to traditional ones as it is often claimed, then it is likely that requirements are handled differently and thus play a different role in influencing IT innovation. This essay thus proposes the study of the role of requirements in open source IT innovation.

**Key words:** requirements engineering, requirements, innovation, open source

## 1 Introduction

Requirements are often seen as representing an attempt to define a future IT system. This may sound simple, but as early as Brooks' time, deciding on the requirements was the "hardest, single part of building an IT system" [1]. Since requirements attempt to define IT, it can be supposed that they have some link to IT innovation. Further, requirements in OS seem to be handled differently than traditional methodologies suggest [2], or offer an alternative to software development [3].

To which extent does this hold true, and under what conditions? What is the role of OS requirements IT innovation?

This essay proposes the study of the role of requirements in OS IT innovation. Section 2 presents a brief review of the literature. Section 3 exposes the problem statement. Section 4 presents the conceptual framework, and finally, section 5 proposes a research design.

## 2 Research Background

### 2.1. Traditional Requirements Engineering

Traditionally, requirements are seen as an engineering activity. Requirements Engineering (RE) is an essential activity in the software engineering process. RE is concerned with “a branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems [...] and the relationship of these factors” [4]. Jackson, has a similar definition where he describes two interrelated elements: the machine and the world. The machine is the software, and the world is the context and environment [5, 6]. Since the world is the context and the environment, it also represents the problem domain from which goals can be elicited and with which the machine can interact to achieve some requirements. Hence, the *raison d’être* of any program lies entirely in the problem domain, the world. Understanding and modelling the problem domain is difficult, with some researchers arguing that it is impossible [7].

Requirements are not so much found as they are engineered [8]. This does not mean that requirements are “made up”, but that they need to be elicited with techniques and reasoned about by people from different backgrounds. Thus, they do not have an independent existence, but are in fact socially constructed. Techniques are useful to engineers so that they can elicit requirements by exploring the problem domain; what can be for them a “*terra incognita*”.

Traditional computer science approaches to system development is generally criticised for being positivist in nature and making unfounded assumptions concerning the organisational context [9]. Solutions to problems in the development of systems are typically prescriptive and argue for more engineering techniques and modeling methods [8]. This concern for engineering techniques is historically bounded to a time when programs were seen as simple Turing machines, sequentially computing well defined algorithms in order to solve unambiguous problems. This type of literature views the world as ordered, or rather chaotic in the Greek sense of the word, simply awaiting ordering.

Typical RE processes include four principal activities: requirements elicitation, requirements specification (or modeling) and analysis, negotiating and agreeing on requirements, and finally communicating and documenting requirements [10, 11].

### 2.2. Requirements in OS

The problem is that methodologies can take certain assumptions that can influence the success of the requirements engineering process and thereby, the product itself. The assumptions can stem from the epistemological perspective taken by the field in order to simplify the “messiness” of the ambiguous world [12]. One such supposition concerns the organisational context in which methods are

prescribed. The fact that information systems can drift shows the volatility of modeling the world and the importance of the context in which they are put to use [7]. Can it really be assumed that the same method will yield the same causal results in two different contexts? Existing literature also would seem to deny this [2, 13, 14].

Open source would certainly seem to present some interesting differences. The history and culture is largely rooted in academia where the idea of cooperation and openness was strong [15, 16]. From culture, this form of viewing software transformed into an ideology, where the freedom of access to the code and modification is in fact a moral choice or right [16, 17]. These traits are enforced by an open license [17], and appear to allow for widespread collaboration without the need for formal ties. Shared, then, amongst all OS projects is a generic hacker culture [15], that lives along the particular culture latent in any OS project, principally formed up by its community [2].

The community in OS can form an integral part in the development of IT systems innovation, but also in the proposition and elicitation of requirements. Indeed, by effectively allowing anyone, including users, to make changes to software, the project can respond to an increasing heterogeneity of needs and demand [18]. Interestingly, users are not passive anymore, but rather have the potential of becoming full fledged actors in the creation of IT innovation. Thus, the success of an OS project will, at least partly, reveal the market potential that exists for that type of IT system since OS effort is partly borne by users who can also be developers themselves [15]. Thus, part of the market and requirements research appears to be done “on the fly”, or post-hoc, by users of the system themselves proposing functionality [19].

It is often assumed that the source of requirements in OS is a developer’s scratch (Raymond, 2001), and that there seemingly is no project planning [13]. A requirement thus comes from a particular need, and its subsequent development responds to the specific need and motivation of a contributor or user, and not necessarily of a market as would a commercial application [18]. If the particular need of a person is shared with others, then there is a likelihood that the project will gain development impetus and interest [15]. In this sense, OS projects go through a ‘natural selection’ process. Other sources of requirements are bug reports, change requests and feature enhancement from mailing lists, wikis and usenet newsgroups.

There are other sources of requirements. German identifies five for the GNOME project: project leaders, reference applications, discussions through, for example, mailing lists, prototypes and post-hoc requirements when developers who wish a final feature to be included in the release [20].

In terms of specification, requirements appear to be defined by informal descriptions of the system-to-be and are specified, analysed and documented in terms of narratives through online “webs of discourse” [2]. These webs of discourse are

trails of discussion concerning the informal requirements, and which help making sense of what it is that should be implemented.

Another consequence of such openness is the ability for large scale peer reviewing, analysis and negotiation of requirements. In terms of bugs, peer reviewing with just one more person drastically increases the reliability of the code [15]. For this reason, many agile development methods advice the use of peer reviewing. Open source here goes further. By allowing the development artefacts and the code to be opened to anyone, peer reviewing can be done at a larger scale, leading to Raymond's Linus' law: "the more eyeballs, the less bugs" [15]. Could this also be true for requirements? If the more people implicate themselves actively, then can the requirements process end up with more accurate requirements? Also, the fact that the source code is open, that contributions and peer reviewing is encouraged, work on open source projects also appear to be, in many cases, highly decentralised collaborations [21].

Whether a request is actually carried out depends on the OS project, and the process associated with the selection of requirements. In Apache, developers (especially core ones) need to be convinced of the usefulness of the feature request [13]. Other OS projects can instead use voting systems. The amount and quality of information that is given by the requestor can also be seen as filters. If key information is missing, then the request has less chances of being considered [13]. Other OS projects such as Mozilla.org, however, also provide a roadmap outlining functionality for future releases [13].

### **2.3. Linking Requirements to IT Innovation**

There are numerous studies on innovation. Some have looked at IT innovation from an organisational perspective [22, 23], while others have looked at user created innovation [24, 25]. Other studies have looked at the innovation process itself, but from an economical and motivational perspective [26-28].

Open innovation advocates for the openness and sharing of innovation which, it is hoped, will increase innovation and the creation of new markets since ideas can be freely exploited by anyone without major legal constraints [26, 27]. Moreover, if one believes that the diffusion of innovation stems from imitation [24], then the fact that the code, and therefore, requirements and functionality is open and free to be copied and reused, encourages innovation.

Open innovation is not necessarily synonymous with open source. It could very well be used by traditional organisations. But the similarities between the OS organisational characteristics, and this innovation model are numerous. Indeed, no impediment to copying, to redistributing or to building upon existing innovation. The only constraint that some OS licenses might impose is the necessity to contribute improvements or innovations back to the community [29]. For OS projects, the community would form an integral part in the development of IT systems innovation.

Indeed, it is one of the proposed reasons as to why open innovation, by effectively allowing anyone, including users, to make changes to software can respond to an increasing heterogeneity of demand [18].

These meta-models look at innovation from an economical and organisational point of view, and in so doing can lose sight of the IT artefact [30]. IT itself is more than a simple black-boxed entity; it can be quite complex, especially in the process of IT creation. The subprocess of Requirements Engineering, is composed of actors - stakeholders, physical artefacts (specifications, design models, etc...), but also of implicit or hidden desires and knowledge that can be expressed at any moment of the software development process. Further, if requirements define the functionality and properties of an IT system, could the management of requirements have repercussions on IT innovation?

## 2.4. Discussion

This essay has presented a review of software development and the requirements process as being polarised between traditional software engineering and open source software. However, the differences might be much greyer in practice. It seems important now to mitigate this polarisation.

Some influential computer science literature, based on Agile methodologies such as eXtreme Programming, in fact, assumes that change is inevitable, and thus that prescribed techniques must be flexible enough to reposition the definition of the software (requirements for developers) to the most current and correct understanding [31]. Although it is also based upon prescriptive indications, this stream of literature in software engineering has recognised the complexity and ambiguity of an ever changing world and proposes ways to reduce complexity and increase flexibility by making, say, requirements more informal.

Another critique of this polarisation could come from a recent paper that claims that open source software is becoming more like its closed source opposite and vice versa [32]. Also, even though positivist literature abounds with examples of methods and claims of good practice, it seems that they are mostly ideals [33], that either need to be faked purposely [34], or altogether unattainable.

But what is open source? And for that matter, what is a traditional organisation? These are quite complex questions, and some aspects often described as open source development can be found in traditional organisations as well. The existence of a strong community also exists outside open source. Apple and the App Store, for example, have created an important crowd sourcing environment with the creation of applications that can meet a wide variety of needs.

Open source projects can also have a strong link to traditional organisations. Mozilla, for example, is described as a hybrid [13]. Do hybrid open source project handle requirements differently than non-hybrids?

Does OS simply refer to the openness of the code and the development artefacts, unconcerned? If this reasoning is followed, then, the only difference

between a closed IT system and an open one is a legal or ideological matter. Open source would therefore closely resemble to crowd sourcing models, but with the source code open. There are, however, some indications from the literature that this openness does bring some changes in the development, practices and management of IT systems [2, 3, 14, 35]. It would be interesting to investigate to which extent this holds true, and under what conditions.

### **3 Problem Statement**

The research question is thus focused on understanding the role that requirements play in OS IT innovation. In this sense, the agenda centers on understanding how requirements are handled within open source projects for the development of IT innovation. The innovation referred to here is not to be understood solely in terms of software, but also in terms of the development practices used. This research induces a second layer of related sub-research questions. Firstly, what is the relationship between requirements and IT innovation. Can IT innovation be promoted or hindered through the management of requirements? Secondly, what are the differences between OSRP and traditional RE? Is OS requirements handled as is claimed by the literature? Are there any exceptions? For example, it is imaginable that requirements in health systems could be handled less informally given the importance of safety and security requirements. Is there some nature of engineering in the OSRP, if so, how different is it compared to traditional RE? Thirdly, how does the “openness” concept influence the requirements process? Is there some typicality in the OSRP that can be attributed to OS characteristics? How does this play in terms of the community, the various users and the possible implication or sponsorship of traditional companies?

The aim of the study is not to edict prescriptive statements of good practice in open source, but rather investigating the situated handling of requirements in their context.

### **4 Theoretical Framework**

Following Crotty’s taxonomy, we take a phenomenological theoretical stance from which to analyse the requirements [36]. Studying the open source requirements process and its relation to IT innovation from a phenomenological perspective allows for the study of how requirements and its processes are dealt with on a day to day basis. As Ciborra says, “the phenomenological perspective can help us recognise and value the importance of bricolage versus the over-inflated role of method and planning in strategic applications” [12, p.21]. The concepts of emergence, bricolage and drifting are put up against attempts to plan, document and specify requirements. This section briefly introduces these concepts.

Hypomnemata is a concept from old Greece and mentioned by Foucault in one of his interviews [37], when the Greeks kept special agendas that were useful to them. These were not used for keeping records of what had happened, but instead for how they might better tackle similar future problems [37]. These agendas had an essential role to play in the organisation of one self and Foucault claims the emergence of this technology as being as disruptive for ancient Greece as the invention of the computer. This material memory, Foucault argues, became an important element for the culture of the self since hypomnemata recorded accounts that could eventually help in the guidance of future actions [37]. In what concerns requirements, these could be specifications, models and documentations which act in the same way for IT innovation by providing guidelines for future actions. Hypomnemata, however, do not need to be fixed documents whose main purpose is one of guidance. It could well be that, for example, the proposal of a feature creates a certain debate in the community, and that once written, the debate will constrain future related features, with say, references by authors to the original discussion. Although this is a hypothetical example, it is reasonable to envisage that there exists heated discussions concerning certain requirements, and since all is made public in OS projects, OS requirements could possess certain characteristics that endure in time. Hypomnemata, therefore, are not necessarily central requirements document, but could also well be previous discussions that could constrain future debates.

On the other hand, the phenomenological concepts drawn from Ciborra of emergence and the related concepts of improvisation, bricolage, drifting, on the contrary, posit that the world cannot be modeled given the complexity of the world [38, 39]. As aforementioned, the concepts can help in understanding the importance of the spontaneous, the unexpected and the messiness of the world. The concepts of emergence, bricolage, tinkering and drifting are closely related. Analysed as moods, instead of mindsets [40], they represent a bottom-up approach to create emergent, and possibly drifting innovation by the understanding of local needs and available resources. The emergent results of bricolage can be surprising, probably because of their bottom-up mentality. Another concept that could help understand the tension between emergence and hypomnemata in context is hospitality [12].

The tension zone between the emergence, drifting and bricolage conflicts with the need to control and direct. However, given the appearing informality in the treatment of requirements in OS, then could it be that the participants, in fact, embrace a methodology of change, with elements of requirements drifting? The requirements handling process could well be seen as a technology in itself, in which case, the process could be accepted, or worked around [41], or if there are attempts to rationalise the requirements process. The informality of requirements therefore leads to the impression that they can have an inchoate aspect with entropic properties. They can easily be changed through discussion, and the purpose of the IT software can be redefined. If, on the other hand, they are written down, with explicit intentions to convince, and ultimately plan for requirements, then these hypomnemata could make the requirements process context more constraining and focused on controlling, coordinating and planning functionality.

Moreover, it is possible, for example, for a requirement to be created spontaneously, and for it to be progressively crystalised, becoming less malleable and achieving a certain hard status through, possibly, collective planning.

## 5 Proposed Research Design

This section presents a brief account of the planned empirical investigation. In summary, the plan consists of explorative, multiple case studies on projects spanning different market sectors. The data collection methods center around electronic archival exploration, supported by interviews that aim to address specific issues as informed by the archival exploration. The research planned should be qualitative so as to study the phenomena in context [42], but the use of quantitative methods are also planned as support.

Case studies are useful in studying requirements in Open Source for several reasons. Firstly, they provide the opportunity to study requirements in a context, as contemporary events, but also in a larger, historical time frame [43, 44]. Also, events surrounding requirements cannot be controlled, which rules out the experimental research method [43, 44]. Given the specificities of OS projects, depending on many contextual factors, such as the market sector or project history or contexts of use, and therefore, the expectation of different results, it would seem that a multiple case study approach is appropriate (Yin 2003). Further, it is hoped that the assumptions on OS software could be tested in various environments, making multiple case studies a desirable approach [43]. The use of case studies in OS research is extensive [2, 13, 20, 45-47].

Four suggested projects are foreseen: Moodle, OpenEHR, OpenOffice.org and BOINC (Berkley Open Infrastructure for Network Computing). Moodle is a widely used educational course management software. OpenEHR is an open specification standard for the management of patient health data in electronic health records. Interestingly, it attempts to model and build a high-level specification of electronic health records that then can be instantiated to context specific clinical environments. It is hoped that the open specification is flexible enough to accommodate differences in business processes, but at the time providing benefits from interoperability and standards. OpenOffice.org is aimed at providing an OS office suite alternative, and is thus intended to fulfill the role of an open competitor to, principally, MS Office. BOINC is a middleware grid computing software.

The projects were chosen according to various criteria. First, it was decided to avoid the typical success stories, such as Mozilla, Linux and Apache, which have already been studied to some extent. Out of the four proposed cases, in fact, only the OpenOffice.org is a mainstream desktop application. The remaining three pertain to some form of niche sector. Following from this, the participants to the projects will likely have different backgrounds, interests and knowledge in IT development. It is probable then that the participants in Moodle will have a strong academic

background, nurses and doctors will be involved in OpenEHR and BOINC of physicists and scientists. The choice of the projects is therefore dictated by their differences rather than by their actual aims.

Regarding data collection, two methods are envisaged: archival exploration and interviews, both appropriate for case studies and allow for construct validity [43, 44].

Archival exploration represents a natural way of studying Open source projects given the general commitment to openness and traceability [15]. There are various sources from which requirements can be elicited: mailing lists, wikis, todo lists, requirements specifications or even bug tracker tools. These provide an unbiased account of the quotidian dealings of requirements. Analysing these allows for the qualification of the construction of requirements and the overall IT innovation process as being either emergent or more focused on planning the overall IT system.

Interviews, on the other hand, could provide second hand accounts on motivation, reasoning and backgrounds which would help the interpretation of the meaning of certain requirements or how they were formulated, or simply help understand the subject area better. Interviews would be semi structured as this provides a good level of flexibility [48]. It would be useful to have a good mix of participants corresponding to the following categories: engaged developer, project governance member, owners of a coding module, non-coding members who participate often, and finally, non-coding members who do not participate often. This would allow to take into account the participants' backgrounds and their possible influence on requirements.

## **6 Conclusion**

The aim of this essay was to present a research proposal concerning the role of requirements in OS IT innovation. This essay first briefly reviewed the research areas involved and presented a theoretical framework to the proposed problem statement. Finally, a research design was proposed.

The expected contributions from this work then are to, firstly, provide a better understanding of the RE process in OS. Secondly, to better understand the relation between the RE and the IT innovation process.

## References

1. Brooks, F.P., *The Mythical Man-Month - Essays on Software Engineering*. 1995: Addison Wesley.
2. Scacchi, W., Understanding the requirements for developing open source software systems, in *IEE Proceedings - Software*. 2002. p. 24--39.
3. Scacchi, W., et al., Understanding Free/Open Source Software Development Processes. *Software Process Improvement and Practice*, 2006. **11**(2): p. 95--105.
4. Zave, P., Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys*, 1997. **29**(4): p. 315--321.
5. Jackson, M., The world and the machine, in *ICSE '95: Proceedings of the 17th international conference on Software engineering*. 1995, ACM. p. 283-292.
6. Jackson, M. and P. Zave, Deriving specifications from requirements: an example. 1995: ACM.
7. Ciborra, C., Dérive, in *The Labyrinths of Information: Challenging the Wisdom of Systems*. 2002, Oxford University Press: Oxford. p. 83--101.
8. Bell, T.E. and T.A. Thayer, Software requirements: Are they really a problem?, in *Proceedings of the 2nd international conference on Software engineering*. 1976, IEEE Computer Society Press: San Francisco, California, United States. p. 61-68.
9. Oates, B. and B. Fitzgerald, Multi-metaphor method: organizational metaphors in information systems development. *Information Systems Journal*, 2007. **17**(4): p. 421-449.
10. Lamsweerde, A.v., Requirements in the Year 2000: A Roadmap, in *22nd International Conference on Software Engineering (ICSE '00)*. 2000, IEEE Computer Society. p. 5.
11. Nuseibeh, B. and S. Easterbrook, Requirements Engineering: A Roadmap, in *Proceedings of International Conference on Software Engineering (ICSE-2000)*. 2000, ACM Press: Limerick, Ireland.
12. Ciborra, C., Encountering Information Systems as a Phenomenon, C. Avgerou, C. Ciborra, and F. Land, Editors. 2004, Oxford University Press. p. 17-37.
13. Mockus, A., R.T. Fielding, and J.D. Herbsleb, Two Case Studies of Open Source Software Development, in *Perspectives on Free and Open Source Software*, J. Feller, et al., Editors. 2005, MIT Press. p. 163-209.
14. Robbins, J., Adopting OSSE Practices by Adopting OSSE Tools, in *Perspectives on Free and Open Source Software*, J. Feller, et al., Editors. 2005, MIT Press. p. 245-264.
15. Raymond, E.S., *The Cathedral and the Bazaar*. 2001: O'Reilly.
16. Stallman, R.M., *The GNU Operating System and the Free Software Movement*. Open Sources: Voices from the Open Source Revolution. 1999: O'Reilly.
17. Perens, B., *Open Sources: Voices from the Open Source Revolution*. 1999.
18. Hippel, E.v. and R. Katz, Shifting Innovation to Users via Toolkits. *Management Science*, 2002. **48**(7): p. 821-833.
19. German, M.D., Software Engineering Practices in GNOME, in *Perspectives on Free and Open Source Software*, J. Feller, et al., Editors. 2006, MIT Press. p. 211-225.
20. German, M.D., Software Engineering Practices in GNOME, in *Perspectives on Free and Open Source Software*, J. Feller, et al., Editors. 2005, MIT Press. p. 211-225.
21. Ljungberg, J., Open source movements as a model for organising. *European Journal of Information Systems*, 2000. **9**(4): p. 208--216.
22. Orlikowski, W., Improvising organizational transformation over time: a situated change perspective. *Information Systems Research*, 1996. **7**(1): p. 63--92.

23. Orlikowski, W., Using Technology and Constituting Structures: A Practice Lense for Studying Technology in Organizations. *Organization Science*, 2000. **11**(4): p. 404--428.
24. Ciborra, C., *Bricolage*. 2002, Oxford University Press: Oxford. p. 29-53.
25. Lin, A. and T. Cornford, *Sociotechnical Perspectives on Emergence Phenomena*. The New Socio Tech. 2000, Surrey, England: Springer-Verlag.
26. Chesbrough, H.W., Open innovation and strategy. *California management review*, 2007. **50**(1): p. 57-76.
27. Chesbrough, H.W., *Open Innovation*. 2003: Harvard Business School Press.
28. Hippel, E.v. and G.v. Krogh, Open Source Software and the ``Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science*, 2003. **14**(2): p. 208-233.
29. Weinstock, C.B. and S.A. Hissam, Making Ligthning Strike Twice, in *Perspectives on Free and Open Source Software*, J. Feller, et al., Editors. 2006, MIT Press. p. 143-159.
30. Orlikowski, W. and C.S. Iacono, Research Commentary: Desperately Seeking the "IT" in IT Research - a Call to Theorizing the IT Artefact. *Information Systems Research*, 2001. **12**(2): p. 121--134.
31. Beck, K., Embracing Change with Extreme Programming. *Computer*, 1999. **32**(10): p. 70-77.
32. Fitzgerald, B., The transformation of open source software. *MISQ*, 2006. **30**(3): p. 587-599.
33. Truex, D., R. Baskerville, and J. Travis, Amethodical systems development: the deferred meaning of systems development methods Accounting, Management and Information Technologies, 2000. **10**(1): p. 53-79.
34. Parnas, D.L. and P.C. Clements, A Rational Process: How and Why to Fake It. *IEEE Transactions on Software Engineering*, 1986. **12**(2): p. 251--256.
35. O'Reilly, T., Lessons from open-source software development. *Communications of the ACM*, 1999. **42**(4): p. 32 -- 37.
36. Crotty, M., *The Foundations of Social Research The Foundations of Social Research - Meaning and Perspective in the Research Process*. 1998: Sage.
37. Foucault, M., An Interview with Michel Foucault, in *The Foucault Reader*, P. Rabinow, Editor. 1984, Pantheon: New York. p. 363--5.
38. Ciborra, C., *The Labyrinths of Information: Challenging the Wisdom of Systems*. 2002: Oxford University Press.
39. Elbanna, A.R., The validity of the improvisation argument in the implementation of rigid technology: the case of ERP systems. *Journal of Information Technology*, 2006. **21**(3): p. 165--175.
40. Weick, K.E., Introductory Essay: Improvisation as a Mindset for Organizational Analysis. *Organization science*, 1998. **9**(5): p. 543-555.
41. Pollock, N., When Is a Work-Around? Conflict and Negotiation in Computer Systems Development. *Science, Technology & Human Values*, 2005. **30**(4): p. 496-514.
42. Silverman, D., *Doing Qualitative Research: A Practical Handbook*. 2005: Sage.
43. Benbasat, I., D.K. Goldstein, and M. Mead, The Case Research Strategy in Studies of Information Systems. *MIS Quarterly*, 1987. **11**(3): p. 369--386.
44. Yin, R.K., *Case Study Research - Design and Methods*. 2003: Sage.
45. Darking, M. and E. Whitley, Towards an Understanding of FLOSS. *Science Studies*, 2007. **20**(2): p. 13-33.
46. Stol, K., et al. The use of empirical methods in Open Source Software research: Facts, trends and future directions. in *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. 2009.

47. Rusovan, S., M. Lawford, and D.L. Parnas, Assessing Open Source Software Development, in Perspectives on Free and Open Source Software, J. Feller, et al., Editors. 2005, MIT Press. p. 107-121.
48. Myers, M.D. and M. Newman, The Qualitative Interview in IS Research: Examining the Craft. *Information and Organization*, 2007. **17**(1): p. 2-26.

# Open Source – Open Community? A Critical Analysis of Stakeholder Integration in Free and Open Source Communities

Celina Raffl

ICT&S Center

University of Salzburg, Austria

<http://www.icts.uni-salzburg.at/raffl>; <http://www.celina.uti.at>

Sigmund Haffner Gasse 18, 5020 Salzburg, Austria

[celina.raffl@sbg.ac.at](mailto:celina.raffl@sbg.ac.at)

**Abstract.** The aim of this PhD project is to analyze the role of professionalization, hierarchies, social exchange, social capital, and information sharing in free and open source (F/OSS) communities. I focus on the research of selected F/OSS projects on two levels: on the one hand I study the participants of F/OSS communities (who they are, what drives their motivation, how they perceive their role within the community) and on the other hand I analyze the community structure according to parameters such as professionalization, communication processes, information sharing, knowledge exchange, influence of hierarchies within the projects, participation of stakeholders, integration of newcomers, etc. Therefore I will develop a framework for analyzing selected F/OSS projects (including small-scale and large-scale projects as well as projects with hired and thus paid developers). The focus of this thesis is on the theoretical discussion of F/OSS as a self-organizing system. An online survey amongst participants of selected F/OSS communities is used to proof the theoretical promises.

**Key words:** communities, free and open source software development, participation, professionalization, stakeholder integration

## 1 Introduction

In recent years free and open source software (F/OSS)<sup>1</sup> that is software that can be run, distributed, shared, studied, changed, and improved by its users has become in

<sup>1</sup> There is an ongoing ideologically biased debate regarding the terms free software [45] and open source [8, 34]. I do respect that there are certain differences regarding the terms. As Scacchi et al. 2006 point out: “[F]ree software is always available as OSS [open source software], but OSS is not always free software” [43]. Since these connotations are not im-

Among them are IT companies or other companies offering technical products or services, governments, public administration, educational institutions, organizations (in particular not-for-profit), hospitals, and private users.

Over the last few years some companies and organizations have based their IT infrastructure on F/OSS for ideological as well as for monetary reasons. Free and open source software is claimed to be reliable, secure, and more up to date than many closed source software; thus F/OSS becomes more and more a valid alternative to many proprietary products [48]. The possibilities of freely sharing and distributing, remixing, adapting, and improving code and content and making them available to others [45], are the main advantages.

These advantages of F/OSS led to a change of the user demography. Early projects were usually invented by a skilled developer to serve his (rather than her) personal needs or to meet the requirements of a small, savvy community. The users of this software were those who invented it. Nowadays there are an increasing number of non-technical (“less skilled”) home computer users and people working in the above-mentioned sectors who use free and open source software<sup>2</sup>.

Furthermore we face a shift towards increasing professionalization within F/OSS, i.e. more involvement of industries and governments, companies that take over successful F/OSS projects, hired and paid developers, etc. Previous studies on the motivation of F/OSS developers [42, 19, 26] stressed the gain of value through learning or becoming acknowledged in the community and thus creating primarily social capital whereas monetary capital was only involved in a second step or indirectly (e.g. after being known in the community getting a well paid job). Current developments such as increasing professionalization let assume that the main interest nowadays (or at least in the near future) focusses primarily on gaining monetary capital. While understandable from an individual developers view (as well as from micro-economics) this paradigm shift could have a major impact on the future direction of F/OSS in particular when considering the original norms and values of the free software movement and the hacker culture as well as the impact F/OSS theoretically could have on societies as formulated by the UN Millennium Development Goals [37].

## 2 Background of the Research

In 2001 the United Nations defined the strengthening of free and open source software as one strategy to reach the Millennium Development Goals [37]. While there

---

portant at this stage of research, I will use the abbreviation F/OSS for free and open source software most of the time, as well as free software when appropriate (e.g. referring to the early days of the movement) or open source since the latter terminology is predominant both in practice as well as in research.

<sup>2</sup> The success of F/OSS as an alternative in an existing software market has been supported by projects such as GNOME or KDE, which emphasize on simplicity and usability of the software.

are many global problems such as starvation, environmental issues, global warming, etc. the United Nations are aware that access to internet and information is crucial for citizens. Knowledge and information are claimed to be the most valuable commodity in current western society able to transform cultural, social and especially economic values [4, 41, 29, 46]. The expression that we are living in an information (or knowledge) society suggests, that knowledge and information are important resources and access is crucial since it enables people to live self-determined lifestyles and to participate in decision-making processes, but limits in access, lack in skills and literacy exclude huge parts of the population from information. Living in an information society means that information has become one, if not the most important resource. Information is commodified and transformed into a scarce resource just like raw materials in industrial nations. This causes gaps between those who have power and access to information and those who are less advantaged. Possessors, creators, and distributors of information thus can decide upon the accessibility of information to others. Its value is artificial because information differs from other (i.e. physical) materials since it can be shared, distributed or copied without value lost, information goods can be shared without losing the possibility of re-using it.

As the United Nations point out, implementing technologies is not an end in itself to solve problems, but providing access and empowering people to use them has to be seen as a tool towards that. ICTs provide the foundation for communities to emerge and to shape society for both societal benefits, e.g. empowerment of citizens, ecological conservation, democratization and participation, as well as negative consequences, e.g. social inequalities, imbalanced power structures, or digital divides [30]. This means that ICTs in general, F/OSS in particular, have the potential to enhance cooperation and their assumed positive effects but they can also foster competition and social segregation. F/OSS depicts on ethical issues like freedom, openness, and inclusion and thus can contribute to the emergence of a cooperative and inclusive society. Hence this research project is embedded in the larger area of information society research, in particular from a normative view questioning how F/OSS can contribute to the positive realization of a future inclusive “Global Sustainable Information Society” [16, 17, 25].

## 2.1 Cooperation and Integration of Stakeholders in F/OSS

Cooperation and the integration of diverse stakeholders and communities are crucial components of free and open source software development. In *The Cathedral and the Bazaar* Eric S. Raymond points out: “Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.” Or more colloquially: ‘given enough eyeballs, all bugs are shallow.’ [40]. This open source dictum refers to an appeal that is user-centered and participative, and suggests the integration of stakeholders in the process of software development. F/OSS development includes a large number of co-developers and users whose perspectives and comments should be considered. Developers expect their community, i.e. other developers and users to react on certain programs or re-

leases. This suggests that a large number of stakeholders are (or at least should be) acknowledged as central resource of knowledge and that they should be treated as valuable information resource.

Integrating stakeholders in the design process can lead to technologies that are more accessible, useful and usable, and which serve users demands but this also bears its challenges. From science-technology studies (STS) view the question of who the relevant stakeholders are, is still problematic and remains unsolved. This issue is addressed as the problem of effective control [33, 39]. In short, the problem of effective control means that the more stakeholders are involved in the project, the more complex and unpredictable it might become. If, on the other hand, less stakeholders are integrated, one risks to cause blind spots in development, or misses certain users' demands. Closely related to this problem is the question when to integrate stakeholders, referring to the problem of timing [33, 39], also known as Collingridge Dilemma (1980) [10]. There is still no common understanding of who should be considered a relevant stakeholder in a F/OSS community and, once identified, when s/he should best be integrated in the development or design process and how much power their decisions (should) have.

## 2.2 Identifying Relevant Stakeholders in F/OSS Communities

One common approach to describe participants or stakeholders in F/OSS is the hierarchical *onion model* [22]. In the centre of the onion is the core team, i.e. the core developer community. These developers are in many cases, especially in large projects, company hired software development staff (up to forty percent according the study of Lakhani and Wolf in 2005 [26]), who get paid to work on F/OSS projects [43].

The core team is respected for its technical capabilities and activities, and thus has the authority to decide upon what will be included in the code base. Usually it is one person (a benevolent dictator) or a small group of core developers that control the architecture and direction of development of the vast majority of F/OSS projects [43]. On the next level we find co-developers. Hedberg and Iivari [22] suggest to distinguish between committers and contributors. Both are active participants in F/OSS projects, but differ in terms of decision-making. Whereas committers can write code and have the power to change the source code directly, contributors send patches and can only hope that they will be integrated in the next release [22, 14]. "Contributors are developers that are users, but not part of the core development team employed by Sun or otherwise nominated and voted into the core development team" [2].

The outer layers of the onion-like F/OSS-model constitute of active and passive users. Whereas the latter are barely recognized in F/OSS research, active users often play a significant role since they are able and willing to participate in F/OSS projects, not directly to the source code, but to other important aspects of the F/OSS environment, e.g. writing bug reports, or answering user requests. Poderi [36] criticizes the marginalized role of users in F/OSS research and suggests to acknowledge

the importance of “peripheral participation in free and open source software communities of practice.”

In a large number of F/OSS projects many end-users participate in and contribute to F/OSS development by providing feedback, writing bug reports, answering requests and raising usability concerns. One must add that even passive users are important for F/OSS projects since maintenance and further development of some projects would probably stop, if there were no people who wanted to use the software. Passive users often recommend software to others and thus work as facilitators of software diffusion and function as advertisers of many projects.

This hierarchical onion-like F/OSS model also suggests to integrate stakeholders in several steps of the development. Following Easons argument that the “design of effective socio-technical systems will depend upon the participation of all relevant stakeholders in the design process” all groups of stakeholders have to be considered important and valuable to the sustainability of F/OSS development. Furthermore some open source software projects have shown evidence that the boundaries between the levels are very often impermeable; thus it is difficult to reach the next level of the onion, and almost impossible to become a core developer (cp. for example Linux, Mozilla, Python, etc.). This, however, can be risky in terms of sustainability of a particular project. Strong hierarchies do not support the long-time existence of a particular open source community, because barriers to access the community increase; without novices joining the community and more informed participants entering a next level within a project, the number of team members in a project shrinks since people drop out of a project for several reasons. Thus it takes some time for those participants that have not been involved in the activities of the community to catch up and develop the project further.

Identifying relevant stakeholders in F/OSS is important to integrate perspectives which might be overseen by a small or heterogenous community. Including the whole developer and co-developer base, volunteers, and users on the other hand can make projects too complex to be handled. Selected F/OSS projects will be analyzed according to parameters such as the community size, which types of participants and at which stage of the development process they are involved. This analysis can contribute to gain knowledge about design approaches that include stakeholders<sup>3</sup>, in order to find out which innovation processes support the liveability and sustainability of a community. An important element in analyzing open source communities and the integration of stakeholders are the participants of the F/OSS projects, their motivation for contribution, and how they perceive issues such as trust, communication processes, the role of information sharing, etc.

<sup>3</sup> This includes approaches such as user-centred design [47, 3, 9, 13], participatory design [44, 6]. Related, concepts are commons-based peer production [5], user innovation [23, 35], collaborative innovation networks [20], collective invention [1, 32], or open innovation [7, 15].

### 2.3 Working Practice in F/OSS Development

A number of studies and research projects have questioned the success of open source software and analyzed the motivation of developers in free and open source software projects. Drivers for the contributions and motivations of open source developers and hackers are for example the enjoyment of learning, to satisfy user needs, to develop skills and to increase the chance for a better job (as a driver for their career), to be recognized in the community (social facilitation) and the believe that software as an information good should be free and available [42, 19, 26] To many, the success of F/OSS appears to be contradictory since developers work for free, and spend their spare time creating something for themselves and give it away for free to a rather unknown or unspecified community. From an economic perspective it seems surprisingly that F/OSS works at all. Eric von Hippel discusses F/OSS as an example for innovation networks and points out, that they do not only exist: “they even triumph!” [23]. According the study of Lakhani and Wolf (2005) [26] about forty percent of the open source developers are (at least part-time) employed.

Traditionally open source developers ‘searched’ for tasks they were able and willing to fulfill – dependent on their skills, talents, and interests. Firm driven open source production aims at reaching certain goals within a given period of time (e.g. for periodical releases). Therefore we need to be better informed about the work environment of paid open source developers, how it is structured, how it differs from those who work without a salary (regarding work flow, ambition, motivation, enthusiasm, work assignment, etc.) and which impact this has on the communities. To ask this question a bit more provocative: If regularly employed open source developers are working in privately organized companies and follow ordinary work instructions, which freedoms remain in free and open source software development? Is free access to the source code all that distinguishes F/OSS from proprietary software? Which implications would this have on the future of F/OSS? By answering these questions this PhD project aims at learning more about information sharing, sustainability of communities and their communication patterns, and conditions that improve participative behavior and sharing of social capital in software development.

## 3 Contribution to the Scientific Discourse

In order to assess and direct the future of F/OSS one has to critically analyze and understand stakeholders and current communities. Therefore this PhD project undertakes research on F/OSS projects on two levels: First we need to understand participants or stakeholders of F/OSS communities (who they are, what drives their motivation, how they consider their role within the community, how they perceive hierarchies, etc.). Second, we have to find out about the community structures. Based on the deeper understanding of the diverse stakeholders involved in F/OSS communities (e.g. core, periphery, users), I will analyze the community structure including communication processes, the way information is shared, and knowledge exchanged. I

seek to find out how stakeholders in F/OSS communities understand their own role, how and if they participate, how open and respectful communities welcome newcomers and how they integrate them. While most of this information can be drawn from theoretical and empirical work that is already available and needs to be rediscussed and recombined under the premises of this thesis, I will further investigate on the matter of professionalization of F/OSS communities and its potential impact on the future of F/OSS development as a movement. Thereby I need to find out more about the role hierarchies and centralization [11] play in F/OSS projects.

From a systems theoretical view hierarchies per se are nothing negative, in contrary, they are important for the system in order to sustain, whereas other parts of the system are in a permanent flux. The realization of hierarchies in working practice however is somewhat different: strong hierarchies in companies usually decrease the free flow of information, increase separation and make it difficult if not impossible for individuals to understand the overall aim of a project or a common goal. Therefore coordination of team members becomes impractical and a collective mind of the team may not emerge [12]. Furthermore, as regards the sustainability of a certain project such a development could be risky since strong hierarchies hinder the long-time existence of a particular open source community, because barriers to access the community increase.

### 3.1 Research Objectives

Taking into consideration the current paradigm shift of F/OSS, we need to find out more about the role of professionalization in order to be able to make a well-informed assessment of the future impact this change has on F/OSS. From this discussion we can draw following propositions:

**Proposition 1a:** Strong hierarchies are important to handle the complexity of F/OSS projects.

**Proposition 1b:** Strong hierarchies within a certain F/OSS project hinder the long-term existence and sustainability of the community.

**Proposition 2a:** If there are too many stakeholders involved the projects becomes too complex to be handled.

**Proposition 2b:** If there are less stakeholders involved the projects one risks to cause blind spots and to miss users' demands.

**Proposition 3a:** If F/OSS working practice functions according the current dominant market logic there is no difference compared to the working practice in other firms.

**Proposition 3b:** For those developers getting paid to work on certain projects the aspects of "freedom" and "openness" have no value (anymore). In other words: Working on open or closed software does not make a difference for those developers.

### 3.2 Research Questions

The overall dominance of the market logic challenges the original norms and values of the free software movement and hacker culture. This leads to following questions:

- RQ 1:** Which impact has the increasing professionalization of F/OSS development on future F/OSS communities?
- RQ 2:** What can we learn from F/OSS communities about information sharing, sustainability of communities and their communication patterns, and conditions that improve participative behavior and sharing of social capital?
- RQ 3:** Is free access to the source code all that distinguishes F/OSS from proprietary software? Which implications would this have on the future of F/OSS, in particular considering the importance of F/OSS as stressed by the UN Millennium Development Goals?

## 4 Research Design

I suggest following methods for further investigation in this research project:

1. development of a conceptual framework based on social self-organization, including the results of related theoretical approaches;
2. online survey among participants in a F/OSS projects, including small-scale and large-scale projects, as well as projects with hired and thus paid developers as a proof of theory.

### 4.1 Theoretical Background

This PhD project discusses the question of openness of F/OSS communities and the influence of hierarchies and centralization based on the theory of self-organization of techno-social systems [18], i.e. evolutionary systems theory [38]. This theory seems appropriate for the course of this work since it includes both fluctuation, chaos, and change as well as the emergence of hierarchies as important factors open systems require in order to sustain. Special emphasize is given to the aspect of professionalization of F/OSS development.

Generally spoken self-organization refers to the capability of any complex open system to create order, i.e. structures, through interactions and interdependencies of the system's parts. The result of this process cannot be fully assessed as well as it is not possible to create the same results when repeating the process – even if the pre-conditions are the same [24]. The order that emerges through the interaction of the system's parts has a new quality that cannot be found in any of the elements of the system; thus we speak of an *emergent phenomenon*. For my thesis I will focus on the systemic and evolutionary character of free and open source software as a dynamic techno-social system. The theory of social self-organization is a meta-theoretical

approach that allows to integrate diverse perspectives that are needed to discuss and explain related issues for this topic (such as exchange theories, coordination theory, participatory culture, participatory design, science-technology-studies, computer supported collaborative work, social capital, social structure, etc.). I need these approaches for identifying relevant variables for the development of a conceptual framework and the online survey I will carry out within selected F/OSS projects.

## 4.2 Case Selection

A sample of relevant F/OSS projects has to be selected for further investigation according to predefined parameters. I will use following projects for the analysis: OpenOffice, Firefox, GIMP, Apache, MySQL. These projects differ in terms of community size, their history, structures, etc. The selected projects need not be comparable since the objective of this PhD project is not to compare the projects but to reflect on a meta-level to answer the above mentioned questions; thus the diversity of the projects should allow to draw a broader picture. What they have in common is that there is already literature available. Generally, in open source research online sources are more valid than in other research areas since the communities are keen to have all material available and accurate, and to provide complete information. Besides the source code there are discussions on mailing lists, boards, forums or chat rooms stored, archived, and freely accessible.

## 4.3 Online Survey

The online survey should go beyond traditional F/OSS questionnaires that ask for the motivation of developers, but seeks to find out about the hierarchy structures in selected projects, perceived differences between paid developers and volunteers, communication processes and information flows. Furthermore this survey should find out whether the theoretical concepts or assumptions meet the real working practice. The survey will consist merely of closed questions. I use the related theoretical approaches to generate questions.

Following criteria are considered relevant from the current point of view and will be used to formulate questions for online surveys to be carried out in the projects as mentioned in section 4.2.

**Size and structure of the community:** Who are the relevant stakeholders? When are they included, at which stage of development are they integrated? Which stakeholders are included?

**Communication structure:** To what extend do opinions of stakeholders matter, how are they acknowledged? Are they considered valuable? How is the commitment of participants to carry out certain duties? Do they have the feeling of belonging to a community? What are their attitude regarding information sharing?

How do they perceive hierarchies? Are they considered important or a constraints?

**Working environment:** Are there differences between those who belong to a certain group within the community (i.e. the layers of the onion)? What are the main differences between those participants (mainly core-developers) that are paid for their work vs. those who spend their free-time contributing to the project?

The PhD project will conclude with a summary and discussion of the results of the empirical part. These findings will iteratively be discussed based on the theoretical approaches (as grounded theory). Potentials for future research in this area will be outlined.

## 5 Expected Outcome

The aim of this PhD project is to analyze the role of professionalization of F/OSS development and its assumed impact on the future of the movement, in particular under consideration of the impact F/OSS has on the information society. Thereby aspects such as hierarchies and centralization [11], social exchange, social capital, and information sharing [21] in F/OSS communities need to be evaluated.

The results of the project can be used to learn from well functioning online communities about other areas where people work together in online environments, e.g. for online or distance learning, for teams with distributed members, for transdisciplinary research teams, etc. Thus it can provide interesting insights to team work in computer-mediated environments, in particular for computer supported collaborative work.

## References

1. Allen, R.C.: Collective invention. *Journal of Economic Behaviour and Organisation* 4(1), 1–24 (1983)
2. Bach, P.M., Carroll, J.M.: Floss ux design. an analysis of user experience design in firefox and openoffice.org. In: C. Boldyreff, K. Crowston, B. Lundell, A.I. Wassermann (eds.) *Open Source Ecosystems. Open Source – Open Community? Diverse Communities Interacting. Proceedings of the 5th IFIP WG 2.13 International Conference on Open Source Systemss, OSS 2009, Skoevde, Sweden, June 2009*, pp. 237 – 250 (2009)
3. Barcellini, F., D’etienne, F., Burkhardt, J.M.: Users’ participation to the design process in an open source software online community (2006). URL <http://arxiv.org/pdf/cs/0612009>
4. Bell, D.: *The Coming of Post-Industrial Society*. Penguin, Harmondsworth (1973)
5. Benkler, Y.: *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, New Haven and London (2006)
6. Bora, A., Hausendorf, H.: Participatory science governance revisited: Normative expectations versus empirical evidence. *Science and Public Policy* 33(7), 478–488 (2006)

7. Chesbrough, H.W.: *Open Innovation: The new imperative for creating and profiting from technology*. Harvard Business School Press (2006)
8. Coar, K.: *The open source definition* (2006). URL <http://www.open-source.org/docs/osd>
9. Collette, N., Julio, A. (eds.): *Inclusive Design Guidelines for HCI*. Taylor & Francis. London, New York. (2001)
10. Collingridge, D.: *The Social Control of Technology*. Frances Pinter: London/New York (1980)
11. Crowston, K., Howison, J.: Hierarchy and centralization in free and open source software team communications. *Knowledge, Technology and Policy* 18, 65–85 (2006)
12. Crowston, K., Kammerer, E.E.: Coordination and collective mind in software requirements development. *IBM Systems Journal* 32(2) (1998)
13. Détienne, F.: Collaborative design: Managing task interdependencies and multiple perspectives. *Interacting with Computers* 18(1), 1–20 (2006). URL <http://dx.doi.org/10.1016/j.intcom.2005.05.001>
14. Divitini, M., Jaccheri, L., Monteiro, E., Traetteberg, H.: Open source process: No place for politics? In: *Proceedings of the 3rd workshop on open source software engineering* Portland, OR, pp. 39–44 (2003)
15. Drossou, O., Kreml, S., Poltermann, A.: *Die wunderbare Wissensvermehrung: Wie Open Innovation unsere Welt revolutioniert*. Heise Verlag: Hannover, Heidelberg (2006). URL <http://www.librarything.de/work/details/26224748>.
16. Fuchs, C.: Co-operation and self-organization. *TripleC. Open Access Journal for a Global Sustainable Information Society* 1(1), 1–52 (2003)
17. Fuchs, C.: Towards a global sustainable information society (GSIS)? *TripleC. Open Access Journal for a Global Sustainable Information Society* 1, 40–99 (2006)
18. Fuchs, C.: *Internet and Society: Social Theory in the Information Age*. Routledge Research in Information Technology and Society. Routledge (2008)
19. Ghosh, R.A., Glott, R., Krieger, B., Robles, G.: *Free/libre and open source software: Survey and study*. floss deliverable final report. Tech. Rep. 4, Survey of Developers, Maastricht (2002)
20. Gloor, P.: *Swarm Creativity. Competitive Advantage Through Collaborative Innovation Networks*. Oxford University Press: Oxford (2006)
21. Hall, H., Widen-Wulff, G.: Social exchange, social capital and information sharing in online environments: lessons from three case studies. paper presented at use. In: *From information provision to knowledge production*, Oulu, Finland (2008). URL <http://www.soc.napier.ac.uk/binary/publication/12035684>
22. Hedberg, H., Iivari, N.: Integrating HCI specialists into open source software development projects. In: C. Boldyreff, K. Crowston, B. Lundell, A.I. Wassermann (eds.) *Open Source Ecosystems. Diverse Communities Interacting*. Proceedings of the 5th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2009, Skoevde, Sweden, June 2009, pp. 251 – 263 (2009)
23. Hippel von, E.: *Democratizing Innovation*. MIT Press: Cambridge, MA (2005)
24. Hofkirchner, W.: *Projekt Eine Welt: Kognition – Kommunikation – Kooperation. Versuch über die Selbstorganisation der Informationsgesellschaft (Reihe Technikphilosophie, Band 9)*. LIT-Verlag: Münster, Hamburg, London (2002)
25. Hofkirchner, W., Fuchs, C., Raffl, C., Schafrank, M., Sandoval, M., Bichler, R.: *ICTs and society – the salzburg approach. towards a theory for, about and by means of the information society*. ICT&S Center: Salzburg 3 (2007). URL <http://icts.sbg.ac.at/media/pdf/pdf1490.pdf>

26. Lakhani, K.R., Wolf, R.G.: Perspectives on Free and Open Source Software, chap. Why Hackers Do What They Do. Understanding Motivation and Effort in Free/Open Source Software Projects. MIT Press: Cambridge, MA (2005)
27. Leavitt, M.O., Shneiderman, B.: Research-Based Web Design & Usability Guidelines. U.S. Government Printing Office (2006)
28. Muffatto, M.: Open Source. A Multidisciplinary Approach. Imperial College Press: London (2006)
29. Negroponte, N.: Being Digital. Coronet, London (1995)
30. Neumayer, C., Raffl, C.: Facebook for global protest. the potential and limits of social software for grassroots activism. In: L. Stillman, G. Johanson (eds.) Proceedings of the 5th Prato Community Informatics & Development Informatics Conference 2008: ICTs for Social Inclusion: What is the Reality? Faculty of Information Technology. Monash University, Caulfield East Australia (2009)
31. Nichols, D., Twidale, M.: The usability of open source software. First Monday. Peer-Reviewed Journal on the Internet 8(1) (2003). URL <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/1018/939>.
32. Osterloh, M., Rota, S., Lüthi, R.: Die wunderbare Wissensvermehrung. Wie Open Innovation unsere Welt revolutioniert., chap. Collective Invention' als neues Innovationsmodell?, pp. 65–76. Heise Verlag: Hannover (2006)
33. Paschen, H., Petermann, T.: Technikfolgenabschätzung als Technikforschung und Politikberatung, chap. Technikfolgenabschätzung – ein strategisches Rahmen-konzept für die Analyse und Bewertung von Technikfolgen, pp. 19–41. Campus-Verlag. Frankfurt/New York (1991)
34. Perens, B.: Open Sources: Voices from the Open Source Revolution, chap. The Open Source Definition. O'Reilly: Richmond, MA. (1999)
35. Piller, F.T.: Die wunderbare Wissensvermehrung. Wie Open Innovation unsere Welt revolutioniert, chap. User Innovation. Der Kunde kann's besser, pp. 85 – 97. Heise Verlag: Hannover (2006)
36. Poderi, G.: Legitimate peripheral participation in free and open source software communities of practice – even non developers enter the community. In: W. Scacchi, K. Ven, J. Verelst (eds.) Proceedings of the Doctoral Consortium of the 5th International Conference on Open Source Systems (OSS2009). June 3-6, Skoevde, Sweden, pp. 73–82 (2009) 37.
37. Portal, U.F.: Free open source software portal. an gateway to resources related to free software and open source technology movement. URL <http://www.unesco-ci.org/cgi-bin/portals/foss/page.cgi>
38. Raffl, C., Hofkirchner, W., Fuchs, C., Schafranek, M.: Cybernetics and Systems 2008, chap. The Web as Techno-Social System. The Emergence of Web 3.0., pp. 604–609. Austrian Society for Cybernetic Studies, Vienna (2008)
39. Rammert, W.: Technik aus soziologischer Perspektive (Band 2). Kultur, Innovation, Virtualität. Westdeutscher Verlag. Wiesbaden (2000)
40. Raymond, E.S.: The Cathedral and the Bazaar. Musings On Linux And Open Source By An Accidental Revolutionary. Sebastopol, CA (1999)
41. Rheingold, H.: The Virtual Community. Homesteading on the Electronic Frontier. Addison-Wesley, Readgin, MA (1993)
42. Robles, G., Scheider, H., Tretkowski, I., Weber, N.: (2001). URL <http://widi.berlios.de/paper/study.html>.

43. Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S.A., Lakhani, K.R.: Guest editorial understanding free/open source software development processes. *Software Process Improvement and Practice* 11, 95 – 105 (2006)
44. Schuler, D., Namioka, A.: *Participatory Design: Principles and Practices*, Lawrence Erlbaum Associate (1993)
45. Stallman, R.M.: *Free Software, Free Society: Selected Essays of Richard M. Stallman*. GNU Press (2002)
46. Toffler, A., Toffler, H.: *Creating a New Civilization. The Politics of the Third Wave*. Turner Publications, Atlanta, GA (1995)
47. Vredenburg, K., Isensee, S., Righi, C.: *User-Centered Design: An Integrated Approach (Software Quality Institute Series)*. Prentice Hall International (2002)
48. Wheeler, D.A.: Why open source software/free software (oss/fs, floss, or foss)? look at the numbers! (2007). URL [http://www.dwheeler.com/oss\\_fs\\_why.html](http://www.dwheeler.com/oss_fs_why.html)



# The Impact of Intellectual Property Enforcement on Open Source Software Adoption

Wen Wen, Chris Forman, and Stuart Graham

Georgia Institute of Technology  
800 West Peachtree Street NW, Atlanta, GA, U.S.A.  
{wen.wen,chris.forman,stuart.graham}@mgt.gatech.edu

**Abstract.** Our research seeks to provide empirical evidence about the impact of intellectual property (IP) enforcement on the patterns of open source software (OSS) adoption. We choose three prominent cases of IP enforcement against OSS sponsors and users – *SCO v. IBM*, *FireStar/DataTern v. Red Hat*, and *Software Tree v. Red Hat*, *HP*, *Genuitec*, and *Dell* to examine whether OSS adoption declines in response to increases in the perceived litigation risk after these events and how will this reaction evolve over time. We further examine the characteristics of OSS and OSS sponsors for which the increase in perceived litigation risk will be greatest.

**Key words:** open source software (OSS), intellectual property enforcement, OSS adoption, litigation risk.

## 1 Introduction

Open source software (OSS) has seen increasing adoption by both firms and individuals in recent years, and in some markets it provides the infrastructure for a significant share of overall economic activity. For example, based on an October 2009 survey by Netcraft, Apache, the most popular open source web server, is responsible for more than 60% of that month's total growth of 4.3 million websites. One key feature of OSS is the OSS user's ability to modify source code based on heterogeneous needs, a characteristic related to the idea of user innovation [1, 2]. Therefore, software firms, especially for start-ups, can compensate for weak internal resources by accessing external resources available in the open source community [3]. Also, they can create value by providing complementary services, releasing code for profits from complementary segments, and organizing OSS development activities [4].

In correspondence to the widely available OSS, how to protect intellectual property (IP) rights of OSS has been emphasized recently, as we see from an increasing variety of licenses under which OSS is distributed and a shift of research interest toward the determinants of OSS license choice [5]. However, there has been little systematic empirical evidence to investigate the impact of IP litigation on OSS

adoption. There are several reasons why this issue is important both to open source developers and users [6]. First, it is costly, if not impossible, to trace OSS back to its origin. This innate problem may increase the likelihood that developers and users of OSS will be a target of enforcement by IP rights holders. Meanwhile, because many sponsors of OSS projects have neither revenue streams nor IP assets of their own, the open source community may be less likely to have the capacity to reduce litigation risks and even to recognize the existence of any legal threats. Furthermore, the licenses under which OSS is distributed usually do not provide warranties sufficient to protect adopters from IP enforcement [6]. These features are likely to slow OSS adoption and inhibit the growth of OSS as a viable alternative to traditional models of developing and commercializing software. Still, a recent survey of young software firms finds that two-thirds are using OSS.<sup>1</sup> Therefore, understanding how IP enforcement shapes OSS user's adoption behavior is important for the future of the OSS movement, the development and growth of new enterprises, and firms' strategic participation in OSS development.

Motivated by these observations, we take a first step toward addressing this issue by providing empirical evidence on the impact of IP enforcement on patterns of OSS adoption. In this study, we are particularly interested in answering two questions. Do open source users negatively react to IP enforcement by decreasing their adoption of OSS? If so, when will their sensitivity to potential legal risks be greatest and how will their reaction evolve over time?

Our focus of this paper is to examine large, well-publicized IP enforcement lawsuits and examine whether the incidence of these lawsuits shifts the expected costs of OSS due to potential legal threat. Based on a search of major news outlets to determine the significance of lawsuits, we choose *SCO v. IBM, FireStar/DataTern v. Red Hat*, and *Software Tree v. Red Hat, HP, Genuitec, and Dell* as three cases to focus upon. Because they happened in different time periods, they help us to understand how the reaction of the open source community to IP infringement risks evolved over time. We plan to use two major data sources. The first and primary data source is SourceForge, which includes over 230,000 projects and over 3 million registered users. However, we notice that some large vendors, such as Red Hat, establish their own web hosts for OSS projects instead of relying on SourceForge. Therefore, we plan to obtain additional data from other sources such as DistroWatch ([www.DistroWatch.com](http://www.DistroWatch.com)), a distribution site summarizing monthly hits for the top 100 most popular Linux distributions.

We have done a preliminary analysis that investigates the relationship between the filing of *SCO v. IBM* and OSS adoption by drawing on data from SourceForge. We have the following results. Basically, IP enforcement is associated with a 45% decline in downloads of OSS projects in the months immediately following the enforcement action. In addition, Linux-kernel OSS projects are faced with an additional 15% greater decline when compared with non-Linux-kernel OSS projects. This is consistent with the hypothesis that Linux-related projects will face the greatest increase in risks from the SCO action, which focused upon IP ownership of

<sup>1</sup> Stuart Graham's unpublished results from the Berkeley Patent Survey (2008)

Linux. Second, as the installed base of the OSS project increases by one standard deviation, downloads decline by an additional 24%. This also confirms our hypothesis that when an OSS project has accumulated a larger installed base, both the likelihood of litigation and potential litigation costs faced by OSS users will be larger. Third, we found firm-sponsored projects experience less of a decline than other projects. This suggests firms' positive role in reducing litigation costs, although they seem to provide more inviting target to the IP rights holder.

## 2 Related Literature

This study would contribute to the following four related fields. First, we contribute to the large literature that examines the relationship between IP protection and innovation [7-10]. While the body of research in this area is large, there is less understanding about how IP rights protection and IP enforcement affect user innovation. We seek to provide the first large scale empirical evidence about the detrimental effect of IP enforcement on one particular form of user innovation – open source innovation. Our primary interest is to understand whether the potential litigation costs from OSS use cause potential users to delay this adoption.

Second, a body of literature has investigated the strategies that firms employ to profit from user innovation [11, 12]. It is worth noting that IP rights substantially shape the way firms interact with the open source community. For example, Henkel [13] shows that firms can protect their IP rights by selectively revealing code to the open source community, while Fosfuri et al. [14] suggest that variations in firms' endowments of IP rights help to determine how they commercialize open source products. We propose to examine how two features of OSS sponsors shape users' response to potential litigation risks from adopting the sponsor's project: the stock of financial and IP resources held by OSS sponsors; the extent to which it promises to indemnify users against the risks of IP litigation. To our knowledge, few studies have examined how firms living symbiotically with the open source community serve as a shield against some IP "terror" shocks. Thus, we intend to show whether firms with different amounts of resources play distinct roles in moderating the potentially negative impact of IP enforcement on OSS adoption.

Third, our study would also add new insights to the literature about information technology (IT) adoption. Researchers have increasingly emphasized the role of external factors on IT adoption such as network externalities (Goolsbee and Klenow 2002; Gowrisankaran and Stavins 2004; Augereau et al. 2006) and local legal regimes [15]. Our study also highlights the role of external factors and seeks to demonstrate how exogenous changes in the perceived risks of IP enforcement influence the costs of adopting IT and whether such costs are perceived to be higher in particular cases.

Fourth, our study has policy implications. It has been observed that the view of the courts toward the validity of software patents has changed radically since 1980, and individual and firm behavior has changed accordingly (Graham and Mowery

[16]). The increasing use of software patents has contributed to a broader concern about the eroding quality of patents [17]. Our study provides further empirical evidence of the rent-seeking strategy made possible by the patentability of software.

### 3 Theory Development

We will first develop a model of OSS adoption which motivates the empirically relevant hypotheses and we then examine whether the empirical evidence is consistent with these hypotheses. Potential adopters decide between adopting a new technology now and deferring this choice for the future by weighting the adoption benefits and costs in an environment characterized by uncertainty [18]. Assuming that benefits remain constant before and after IP enforcement, our economic analysis of the impact of IP enforcement on OSS adoption is mainly based on how IP enforcement changes the expected adoption costs.

Adoption costs for an individual under no IP enforcement is  $C$ ; total adoption is  $D(C) = N_0 \int_C F_B(b) db$  where  $N_0$  is the population size,  $F_B(b)$  is the proportion of potential adopters with valuation  $b$ , and  $\partial D / \partial C < 0$ . In a scenario with IP enforcement, the expected costs for an individual to adopt OSS is  $EC = C + \textit{likelihood} \cdot \textit{litigation costs}$ , where *likelihood* (denoted as  $l$ ) is the litigation risk perceived by this individual that he would be sued and *litigation costs* (denoted as  $LC$ ) are the total costs this individual is faced with to defend himself, and if he loses, to pay for the damages alleged by the plaintiff.

The focus of our study will be to examine how the incidence of large, well-publicized lawsuits such as the *SCO v. IBM* shifts the expected costs of OSS due to potential legal threats. If the potential adopter observes a lawsuit about enforcing IP rights toward the open source community, this would increase the perceived litigation risk of using any OSS, i.e.  $\partial l / \partial \textit{lawsuit} > 0$ . Further, given such a lawsuit, if the alleged ‘infringing’ technology is similar to some OSS in relevant characteristics, the perceived likelihood for that OSS to be detected by the IP rights holder will increase further [19], i.e.  $\partial^2 l / \partial \textit{lawsuit} \partial \textit{similarity} > 0$ . As a result, the extent of adoption declines: we have  $\partial D / \partial \textit{lawsuit} < 0$  and  $\partial^2 D / \partial \textit{lawsuit} \partial \textit{similarity} < 0$ .<sup>2</sup>

One additional opportunity for our research is to examine how the characteristics of OSS projects and project sponsors influence the effects of IP enforcement on adoption behavior. We now focus on two characteristics of OSS projects in particular: the size of the OSS project’s installed base and whether the OSS project is sponsored by firm. Within the set of OSS projects sponsored by firms, we are especially interested in two characteristics of the OSS project sponsor: the stock of

<sup>2</sup> *lawsuit* and *similarity* can be regarded either as a binary or a continuous variable. If *lawsuit* is regarded as a binary variable, it indicates whether a lawsuit is filed; if it is regarded as a continuous variable, it indicates the extent of news coverage of this lawsuit. If *similarity* is regarded as a binary, it indicates whether the technology in dispute is related to the OSS; if it is regarded as a continuous variable, it indicates how similar the technology in dispute is to the OSS in relevant characteristics.

financial and IP resources and whether the sponsor provides any indemnification programs. We expect that these features will shape both the risk (likelihood) and costs of litigation, sometimes in opposing ways.

We first examine the effects of the installed base  $N$  of the ‘infringing’ OSS. We expect  $\partial^2 I / \partial \text{lawsuit} \cdot \partial N > 0$ . This is because when an alleged ‘infringing’ OSS has accumulated a larger installed base, it will generate greater monopoly returns to the IP rights holder if he wins a trial [20, 21]. Thus, because of the greater potential payoffs to the IP enforcer, the more likely he will file a lawsuit. Thus, we have  $\partial^2 D / \partial \text{lawsuit} \cdot \partial N < 0$ .

Another important factor is whether the OSS project is sponsored by a firm or developed by an individual which we measure with a *sponsorship* dummy, denoted as  $S$ .<sup>3</sup> We expect  $\partial LC / \partial S < 0$ : a firm may have greater commercial interactions with the IP rights holder such as cross-license or patent pooling arrangements [19] when compared with an individual, so a firm may help its customers to reduce litigation costs by bargaining with the IP rights holder. Regarding the total number of adoption, therefore, we have  $\partial^2 D / \partial \text{lawsuit} \cdot \partial S > 0$ . At the same time, as we describe in further detail below, it is possible that projects sponsored by firms will also provide a more appealing target because of their greater financial resources, so we have  $\partial^2 I / \partial \text{lawsuit} \cdot \partial S > 0$  and therefore  $\partial^2 D / \partial \text{lawsuit} \cdot \partial S < 0$ . Thus, we hope to empirically identify the dominant one of the two opposite effects of sponsorship.

Within the set of OSS projects sponsored by firms, we can differentiate firm sponsors based on the stock of financial and IP resources (e.g., the number of patents or trademarks). On the one hand, small firms, just as individuals or non-profit organizations, may lack both financial resources and IP resources to help reduce their customers’ litigation costs, i.e. the more resources held by the firm sponsors, the lower litigation costs faced by OSS users and the lower the decline in the OSS adoption. On the other hand, the greater resources of firms will mean that the expected payoff from litigation may be higher. Thus, the IP rights holder may intentionally choose this group as its target, i.e. OSS users may perceive higher likelihood that this group of firm sponsors would be sued. As noted by Mark Webbink, deputy general counsel at Red Hat Inc., ‘the chances of end-users being sued partially depends on the market capitalization of the vendor that provided the open source software.’ [22] Consequently, OSS adoption from this group of firm sponsors will face greater decline. Sorting out these two conflicting effects will be one area of focus for this research.

Another characteristic of sponsor firms is whether the sponsor offers some form of indemnification program  $I$ . One feature of this program we need to notice is that usually indemnification programs are offered by large commercial vendors to large

<sup>3</sup> Another possibility is that the project is sponsored by a private foundation. These foundations may exhibit characteristics of firms and individuals. To the extent that they may not have a patent portfolio they will be more similar to individuals. However, to the extent that they may hold other IP assets such as other licensed OSS projects they may exhibit characteristics of firms. We plan to study the implications of projects sponsored by these types of organizations in our research.

commercial users. Since the program includes “assumption of the complete legal defense (including appeals) and all of the costs associated with that defense in the event that its indemnified customers are sued” [23], we expect OSS users will assume lower litigation costs if they adopt OSS projects with indemnification program. This suggests OSS adoption from this group of firm sponsors will face smaller decline.

## 4 Cases of IP Enforcement

The first prominent case we examine is *SCO v. IBM*. On March 7, 2003, SCO filed a \$1 billion lawsuit against IBM claiming that SCO has the ownership of Unix and all of its derivative works [24]. This lawsuit attracted much public attention: based on a Lexis-Nexis search we found that major news outlets such as the Wall Street Journal, the San Jose Mercury News (Silicon Valley, California), the Boston Globe, the Los Angeles Times, the Daily Telegraph (Sidney, Australia), the Business Times (Singapore) all reported this lawsuit around the filing date among many others. Because it is the first major IP enforcement action between a firm and the open source community, we expect that this case would shift the whole open source community’s prior beliefs about the potential infringement risks of using OSS.

In future work, we plan to examine the effects of two additional suits on open source adoption. The second IP infringement lawsuit is about Red Hat’s JBoss suite of software: *FireStar/DataTern v. Red Hat*. In the late June 2006, FireStar/DataTern filed a lawsuit asserting that JBoss’s Hibernate 3.0 infringes FireStar’s patent which details a method of interfacing an object-oriented software application with a relational database. It was then settled in June 2008 with some permissive terms. Further, as observers noted, this lawsuit ‘is potentially more significant than the SCO case because it’s about a patent that covers a basic concept or idea, not an expression of an idea, which copyright covers.’ [25]

The last IP infringement lawsuit we investigate is also related to Red Hat’s JBoss suite of software: *Software Tree v. Red Hat, HP, Genuitec, and Dell*, which was filed in March 2009. Defendant Red Hat was charged with infringing the patent through its JBoss suite; HP, Genuitec, and Dell were charged with using JBoss-based application. Because it centers on the same technology as *FireStar/DataTern v. Red Hat*, it is interesting to examine whether the open source community, or any particular groups, would react to such similar IP infringement lawsuit again, though *FireStar/DataTern v. Red Hat* had already been settled before.

## 5 Data

### 5.1 Data Sources

We use two major data sources to test the above hypotheses. The first and primary data source, SourceForge, is the largest web host for OSS, where firms and individuals can download and contribute to many kinds of OSS. As of February 2009, SourceForge includes over 230,000 projects and over 3 million registered users [26]. However, restricting our data source to SourceForge may cause us to miss some data on outcomes from important OSS vendors. These large vendors, such as Red Hat, establish their own web hosts for OSS rather than relying on SourceForge (e.g. Red Hat's Fedora and Debian projects). Therefore, we will draw additional data source from DistroWatch ([www.DistroWatch.com](http://www.DistroWatch.com)), a distribution site summarizing monthly hits for the top 100 most popular Linux distributions.

### 5.2 Sample and Variables for Preliminary Analysis

We have already collected data from SourceForge for the first case (*SCO v. IBM*). To control for any year and month effects, the sampling period for this case is from Jan 2002 to Jul 2003, with 14 months before *SCO v. IBM* and 5 months after. To keep the panel balanced, we drop the projects that have any missing observations during this sampling period. Thus, we have 11,536 OSS projects over a period of 19 months. This leads to 219,184 observations in total. The key variables and how we measure them for preliminary analysis are described as follows.

**adoption  $D$**  It is measured by the total number of downloads on SourceForge for project  $i$  in month  $t$ .<sup>4</sup> The unit of analysis for this variable is based on project-month level.

**similarity** We use a dummy to measure whether the “infringing” technology involved in the lawsuit is related to project  $i$  on SourceForge. Because this preliminary analysis focuses on *SCO v. IBM* which centers on Linux, we create a variable “Linux kernel” which will be equal to 1 if the OSS project's topic is related to the Linux kernel and 0 otherwise.

**installed base  $N$**  It is measured by the log transformation of aggregated historical downloads on SourceForge 2 months before the sampling period.

**sponsorship  $S$**  Although we don't have direct measure for this variable, we use an indirect approach based on the homepages of OSS projects on SourceForge. If the project's homepage is in “www.xxx.com” form, we consider them to be the candidates of firm-sponsored projects; for other types of homepages such as “www.xxx.org” or “xxx.sourceforge.net”, we consider them to be projects developed

<sup>4</sup> We admit that using the number of downloads is not a precise measure of adoption, since people may just download OSS project without using it or the same people may download different versions (and thus cause multiple downloads) of one OSS project.

by individual or non-profit organizations. Within the set of OSS with “www.xxx.com”, we further manually identify whether each OSS is sponsored by a firm by retrieving the organization’s name on www.whois.net.

The descriptive statistics of the above major variables are presented in table 1 as follows.

**Table 1.** Descriptive Statistics

	Mean	Std. Dev.	Minimum	Median	Maximum	No. of observations
adoption (downloads)	697.173	9871.871	0	13	1120505	219184 <sup>5</sup>
similarity (Linux kernel)	.012	.108	0	0	1	11536
installed base	4.857	2.961	0	5.280	15.140	11536
sponsorship	.011	.104	0	0	1	11536

### 5.3 Variables for Follow-up Analysis

We are going to investigate additional variables such as the stock of financial and IP resources and provision of indemnification program  $I$ . We will measure each firm’s financial and IP resources by matching the sponsor name on SourceForge with external data sources that record sales and IP assets, such as the USPTO data file and the CorpTech directory of high tech companies. Also, we will identify the indemnification providers by searching their announcements on the major newspapers.

## 6 Empirical Models and Preliminary Results

Beyond the above descriptive analysis, we examine the basic relationship between the filing of *SCO v. IBM* lawsuit (a case of IP enforcement) and OSS adoption by using regression analysis. To control for differences in adoption propensities across OSS projects, we employ a fixed effects model.

First, to test  $\partial D/\partial lawsuit < 0$ , we have the following specification:

$$\log(\text{downloads})_{it} = \alpha \cdot \text{lawsuit}_t + \text{year dummy} + \text{month dummies} + v_i + \varepsilon_{it} \quad (1).$$

<sup>5</sup> adoption (downloads) is measured on project-month level, so it has 219184 observations; similarity, installed base, and sponsorship are measured on project level, so they have 11536 observations.

We use the log transformation of *downloads* as the dependent variable because the distribution of downloads is highly skewed. *lawsuit* is a dummy variable and indicates the presence of any IP infringement lawsuit targeting the open source community.  $v_i$  is a vector of project  $i$ 's time-invariant characteristics, which will be differentiated out by the fixed effect model. Our interest is examining whether  $\alpha < 0$ , which means the significantly negative impact of IP enforcement on OSS adoption. The estimated coefficient  $\alpha$  shown in column (1) in table 2 is significantly negative, suggesting that downloads significantly declined in the months following the filing of *SCO v. IBM*. The magnitude of this coefficient further shows that IP enforcement is associated with a 45% decline in downloads.

Our next step is to investigate which characteristics of OSS projects are associated with the most significant declines in adoption. First, based on our theoretical model, we expect that  $\partial^2 D / \partial \text{lawsuit} \cdot \partial \text{similarity} < 0$ . As mentioned before, for this empirical analysis on *SCO v. IBM* case, we operationalize *similarity* as *Linux kernel*. Therefore, the specification to test this relationship is:

$$\log(\text{downloads})_{it} = \alpha \cdot \text{lawsuit}_t + \beta \cdot \text{lawsuit}_t \cdot \text{Linux kernel}_i + \text{year dummy} + \text{month dummies} + v_i + \varepsilon_{it} \quad (2).$$

Our interest is examining whether  $\alpha < 0$  and also  $\beta < 0$ , which means the significantly negative impact of IP enforcement on OSS adoption and even greater negative impact on Linux kernel projects. As we can see in column (2) in table 2, the estimated coefficient  $\alpha$  and  $\beta$  are both significantly negative. When compared with non-Linux-kernel projects, Linux kernel projects are faced with a 15% greater decline.

Second, our theoretical model suggests  $\partial^2 D / \partial \text{lawsuit} \cdot \partial N < 0$  and this leads to the following specification:

$$\log(\text{downloads})_{it} = \alpha \cdot \text{lawsuit}_t + \beta \cdot \text{lawsuit}_t \cdot \text{Linux kernel}_i + \gamma \cdot \text{lawsuit}_t \cdot \text{installed base}_i + \theta \cdot \text{lawsuit}_t \cdot \text{Linux kernel}_i \cdot \text{installed base}_i + \text{year dummy} + \text{month dummies} + v_i + \varepsilon_{it} \quad (3).$$

The column (3) in table 2 shows the results for this specification. The estimated coefficient  $\gamma$  is -.096. This significant and negative coefficient suggests that as the installed base increases by one standard deviation, IP enforcement is associated with a 24% greater decline in downloads. However, the estimated coefficient  $\beta$  becomes insignificant.

Third, regarding the effect of sponsorship  $S$ , we need to identify the dominant effect from  $\partial^2 I / \partial \text{lawsuit} \cdot \partial S > 0$  and  $\partial^2 LC / \partial \text{lawsuit} \cdot \partial S < 0$ . Thus, we have the following specification:

$$\log(\text{downloads})_{it} = \alpha \cdot \text{lawsuit}_t + \delta \cdot \text{lawsuit}_t \cdot \text{sponsorship}_i + \gamma \cdot \text{lawsuit}_t \cdot \text{installed base}_i + \rho \cdot \text{lawsuit}_t \cdot \text{sponsorship}_i \cdot \text{installed base}_i + \text{year dummy} + \text{month dummies} + v_i + \varepsilon_{it} \quad (4).$$

As we see in column (4) in table 2, the estimated coefficient  $\delta$  is significantly positive and equals to 0.422, suggesting that firm-sponsored projects experience a smaller decline than other projects after the filing of the lawsuit. Furthermore, it is worth noting that the estimated coefficient  $\rho$  of  $lawsuit_t \cdot sponsorship_i \cdot installed\ base_i$  provide further evidence that OSS users react differently to firm-sponsored projects with a large installed base versus firm-sponsored projects with a small installed base.

Last, if we put all variables into one specification (5) as below, the results are shown in the column (5) in table 2. We can see the directions and significance of all variables' coefficients do not change, suggesting the robustness of our results.

$$\log(downloads)_{it} = \alpha \cdot lawsuit_t + \beta \cdot lawsuit_t \cdot Linux\ kernel_i + \delta \cdot lawsuit_t \cdot sponsorship_i + \gamma \cdot lawsuit_t \cdot installed\ base_i + \theta \cdot lawsuit_t \cdot Linux\ kernel_i \cdot installed\ base_i + \rho \cdot lawsuit_t \cdot sponsorship_i \cdot installed\ base_i + year\ dummy + month\ dummies + v_i + \varepsilon_{it} \quad (5).$$

## 7 Conclusions

This study examines the relationship between IP enforcement and OSS adoption. We hypothesize several characteristics of OSS projects and OSS sponsors for which the increase in perceived likelihood of litigation and litigation costs arising from IP enforcement actions will be greatest. The preliminary empirical tests largely support our hypotheses about the detrimental impact of IP enforcement on OSS adoption and how it further interacts with such factors as technology similarity, installed base, and sponsorship.

**Table 2.** Main Results for Specification (1), (2), (3), (4), and (5)

	(1)	(2)	(3)	(4)	(5)
lawsuit	-.600 (.010)**	-.599 (.010)**	-.135 (.012)**	-.140 (.012)**	-.140 (.012)**
lawsuit · Linux kernel		-.160 (.040)**	-.067 (.079)		-.065 (.079)
lawsuit · installed base			-.096 (.001)**	-.095 (.001)**	-.095 (.001)**
lawsuit · Linux kernel · installed base			-.006 (.012)		-.007 (.012)
lawsuit · sponsorship				.422 (.081)**	.423 (.081)**
lawsuit · sponsorship · installed base				-.061 (.012)**	-.061 (.012)**
No. of groups	11536	11536	11536	11536	11536
No. of observations	219184	219184	219184	219184	219184
R square (within)	.264	.264	.279	.279	.279

Notes:

1) +: significant at 10%; \*: significant at 5%; \*\*: significant at 1%.

2) Standard errors are in parentheses

## Reference

1. Von Hippel, E., *Innovation by user communities: Learning from open-source software*. MIT Sloan Management Review, 2001. 42(4): p. 82.
2. Franke, N. and E. Hippel, *Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software*. Research Policy, 2003. 32(7): p. 1199-1215.
3. Stam, W., *When does community participation enhance the performance of open source software companies?* Research Policy, 2009. 38(8): p. 1288-1299.
4. Lerner, J. and J. Tirole, *Some simple economics of open source*. Journal of Industrial Economics, 2002: p. 197-234.
5. Lerner, J. and J. Tirole, *The scope of open source licensing*. Journal of Law, Economics, and Organization, 2005. 21(1): p. 20-56.
6. Buono, F. and M. Sieverding, *Trend Spotting: Recognizing The Growing Risk of IP Litigation Facing OSS Developers And Implementers*, in *Metropolitan Corporate Counsel*. 2008.
7. Heller, M.A. and R.S. Eisenberg, *Can patents deter innovation? The anticommons in biomedical research*. Science, 1998. 280(5364): p. 698.

8. Benkler, Y., *Intellectual property and the organization of information production*. International Review Of Law & Economics, 2002. 22(1): p. 81-107.
9. Murray, F. and S. Stern, *Do formal intellectual property rights hinder the free flow of scientific knowledge? An empirical test of the anti-commons hypothesis*. Journal of Economic Behavior and Organization, 2007. 63(4): p. 648-687.
10. Arora, A., A. Fosfuri, and A. Gambardella, *Markets for technology: The economics of innovation and corporate strategy*. 2001: MIT Press.
11. von Krogh, G. and E. von Hippel, *Special issue on open source software development*. Research Policy, 2003. 32(7): p. 1149-1157.
12. Dahlander, L. and M.G. Magnusson, *Relationships between open source software companies and communities: Observations from Nordic firms*. Research Policy, 2005. 34(4): p. 481-493.
13. Henkel, J., *Selective revealing in open innovation processes: The case of embedded Linux*. Research policy, 2006. 35(7): p. 953-969.
14. Fosfuri, A., M.S. Giarratana, and A. Luzzi, *The penguin has entered the building: The commercialization of open source software products*. Organization Science, 2008. 19(2): p. 292-305.
15. Miller, A.R. and C. Tucker, *Privacy protection and technology diffusion: The case of Electronic Medical Records*. Management Science, 2009. 55(7): p. 1077-1093.
16. Graham, S.J.H. and D.C. Mowery, *Intellectual property protection in the US software industry*. Patents in the Knowledge-based Economy, 2003: p. 219-258.
17. Hall, B.H., *Business method patents, innovation, and policy*. 2003.
18. Hall, B.H. and B. Khan, *Adoption of new technology*. NBER working paper, 2003.
19. Lanjouw, J.O. and M. Schankerman, *Stylized facts of patent litigation: Value, scope and ownership*. 1998, National Bureau of Economic Research Cambridge, Mass., USA.
20. Somaya, D., *Strategic determinants of decisions not to settle patent litigation*. Strategic Management Journal, 2003: p. 17-38.
21. Teece, D.J., *Profiting from technological innovation: Implications for integration, collaboration, licensing and public policy*. Research Policy, 1986. 15(6): p. 285-305.
22. Evers, J., *Do software users need indemnification?* Network World, 2004.
23. Berlind, D. *Novell's protection: Covers more than SCO, caps damages, targets enterprises*. 2004 [cited; Available from: [http://techupdate.zdnet.com/techupdate/stories/main/Novell\\_\\_protection.html](http://techupdate.zdnet.com/techupdate/stories/main/Novell__protection.html)].
24. SCO. *Complaint in the Caldera Systems, Inc. (d/b/a The SCO Group) vs. International Business Machines Corporation case*. 2003 [cited; Available from: <http://sco.tuxrocks.com/Docs/IBM/complaint3.06.03.html>].
25. Rooney, P., *FireStar Files Suit Against Red Hat*, in *COMPUTER RESELLER NEWS*. 2006.
26. SourceForge. *What is SourceForge.net?* 2009 [cited; Available from: <http://sourceforge.net/apps/trac/sourceforge/wiki/What%20is%20SourceForge.net?>].

Part II

## **Lightning Talks**



# Classifying Open-source Project Requirements According to McCall's Model using a Six-level Tagging Grammar

Radu Vlas

Georgia State University, J. Mack Robinson College of Business, Computer Information Systems Department, 35 Broad Street, Room 910, Atlanta, Georgia, USA, rvlas@cis.gsu.edu

**Abstract.** Open source project requirements are typically represented in textual format or by using other informalisms. They are commonly part of postings on forums and discussion boards, email communication among open source participants, or postings of features and bugs requests. The use of informal means of expressing ideas makes the automated analysis of requirements difficult, and thus limits the application of classification, tracing, and other best practices. In this design research study I build, describe, and propose a tag-based grammar method and tool that supports discovery and classification of open source requirements. The tool, implemented in GATE (General Architecture for Text Engineering), relies on a six tag levels model (Requirements Composition Model, or RCM) from general tokens and parts of speech through to McCall's 23 quality criteria. I apply this model and use the associated tool on a number of 10 Source Forge projects selected on their number of downloads (proxy for success), number of developers, and number of features requests. The analysis demonstrates the method and its effectiveness in discovering requirements and in classifying them according to McCall's quality criteria. The evaluation of accuracy of the proposed model requires domain experts to manually discover and classify requirements in sample random fragments of text and then compare their results against tool's output. In assessing the efficiency of the proposed model, I consider a set of measures for discovery (5th level) coverage, and classification (6th level) coverage. I start with an implementation of the 6th level—the classification level—of the RCM relying exclusively on the 23 quality criteria suggested by McCall. I compare this to an implementation which complements McCall's quality criteria with a set of classification keywords and patterns either suggested by Jane Cleland-Huang or recommended by the author. The evaluation results gathered highlight the need to review McCall's quality criteria in order to develop them into a better fit for the current open-source phenomenon. The enhancements proposed by this study improve classification efficiency from an average of almost 30% to an average of almost 65%. These enhancements are coming to suggest a direction for adapting McCall's quality model to the characteristics of current open source environments. A visual inspection of results at the end of the analysis helps conduct a qualitative assessment of efficiency as well as provide informal results concerning the distribution of various types of requirements throughout the lifecycle of the projects.



# Incumbent vs. Challengers in the Office Productivity Software Market

Aaron Baird, T. S. Raghu, and Rajiv Sinha

W.P. Carey School of Business

Arizona State University, Tempe, Arizona

{aaron.baird,raghu.santanam,rajiv.sinha}@asu.edu

WWW Home Page: <http://wpcarey.asu.edu/is/degrees/phdwelcome.cfm>

**Abstract.** Incumbent firms have often not fared well when challenged by low-price innovations. For instance, Microsoft Excel supplanted the dominant Lotus 1-2-3 in the mid-to-late 1980s as the dominant software platform was migrating from the Apple Macintosh to the newly emerging Microsoft Windows (Leibowitz and Margolis 2001). Incumbent firms tend to cater to their most profitable customers and are often blindsided by new entrants with innovative products that cater to the needs of consumers at the opposite end of the market (Christensen 1997). Sheremata (2004) suggests that challengers can effectively compete with entrenched incumbents by adopting high-risk and high-reward strategies that are often unattractive to risk-averse incumbents who already benefit from established network effects. Even more challenging for incumbent proprietary software producers is that many of the more recent competitive pressures come from innovative free alternatives. For instance, Microsoft Office is now threatened by OpenOffice, a free open source software (OSS) software package with very similar features and capabilities, and by Google Docs, a free cloud-computing office productivity suite conveniently available over the Internet. In addition, incumbent proprietary software producers face significant challenges from pirated software. In this paper, we consider the impact of a free/OSS alternative, a free cloud-computing alternative, and software piracy controls on the willingness-to-pay (WTP) and demand for a proprietary software product. We plan to elicit the WTP for a proprietary software product in the presence of these three factors (F/OSS alternative, cloud alternative, piracy controls) in a 2 x 2 x 2 experimental design. We plan to estimate WTP with Double-Bounded Dichotomous Choice (DBDC) Contingent Valuation (CV) survey methods. While many studies have found that challengers have the ability to successfully topple incumbents (e.g. Christen 1997, Sheremata 2004), very few studies have empirically researched the ideas put forth by Hill and Rothaermel (2003) suggesting that incumbents can successfully defend demand in innovative markets. We fill this gap in the literature by empirically analyzing challenges to an incumbent's established network. In contrast to much existing literature on innovation in software markets, we expect to find that F/OSS and cloud-computing alternatives reduce the WTP for the incumbent proprietary software product (Microsoft Office) and increase consumer surplus.



# Improving Open Source Software Patch Contribution Process: Methods and Tools

Bhuricha Deen Sethanandha

Department of Computer Science, Portland State University  
1900 SW 4th Avenue, Portland, OR 97201 USA  
bhuricha@cs.pdx.edu,  
WWW home page: <http://www.cs.pdx.edu/bhuricha>

**Abstract.** Open Source Software (OSS) projects and systems have become significant parts of the software economy. The sustainability of an OSS project depends largely on community contributions; especially patch (source code and document change) contributions. The patch contribution process is very important to OSS projects because it is the primary quality assurance mechanism, enables learning and knowledge transfer in software projects, and provides an opportunity for recruiting developers. Nevertheless, there are several barriers to contribution to mature OSS projects. Patch contribution is time consuming and slow. Patches are often lost, ignored, under-reviewed, or avoidably rejected. Moreover, for contributors participating in multiple projects differences in processes and tools can create confusion and increase learning time. A systematic understanding of the patch contribution process is required in order to determine key problems and improve the process.

The goal of my dissertation work is an improved process for OSS patch contribution. The outcome of this research will be a descriptive model and a set of tools and methods for analyzing and improving the patch contribution process. This dissertation aims to answer three research questions: 1) What are the existing OSS patch contribution processes? 2) What are the underlying problems? 3) How can the process be improved? I am currently conducting a case study on existing OSS projects to understand these issues. The selected projects are mature OSS projects with observable patch contributions. So far, I have conducted a preliminary study and developed a generic OSS patch contribution process model. The model consists of five main activities: patch creation, publication, discovery, review, and, application. I plan to use the model as a foundation to identify key process and tool problems, using the value stream mapping (VSM) technique to analyze the lead time of the process. I will develop enhancements to existing tools such as IDEs, bug tracking systems, and patch tracking systems in order to improve the process by reducing lead time, reducing the number of lost and ignored patches, and increasing the amount of feedback. I will evaluate the tools by conducting experiments to prove that the new tools support an improved patch contribution process. The expected contributions of this dissertation are: 1) to show how to apply lean technique to the patch contribution process, and 2) to provide a set of tools and methods for the OSS communities to support the improved patch contribution process.



# Moving Beyond Continuance Intention toward Loyalty: An Empirical Study in the FLOSS Context

Namjoo Choi

Department of Informatics, University at Albany, SUNY  
7A Harriman, Suite 220, 1400 Washington Ave. Albany, NY 12222, USA  
nc236879@albany.edu

**Abstract.** Recent changes and trends in the IS industry demand an extended view of the IS post-adoption phenomenon that goes beyond mere continued IS use towards loyalty. The advent of the Internet has entailed standardized interfaces, communication protocols and data formats. The proliferation of Free/Libre Open Source Software (FLOSS) offers more choices for users. New features introduced by one provider are quickly replicated by its rivals, and users are becoming more adept at using these standardized applications. In such an environment, one provider's IS product or service can be easily substituted with another's, for example, Internet Explorer (IE) vs. Firefox. Drawing upon the IS post-adoption literature, the literature on FLOSS, and the loyalty literature in both IS and consumer behavior research, this study proposes a theoretical model that extends the horizons of IS post-adoption research to loyalty. More specifically, this study first conceptualizes loyalty in the context of IS research and proposes a new classification of loyalty types: pliant loyalty (loyalty that shows low commitment and thus is subject to switching) vs. rigid loyalty (loyalty that demonstrates high commitment and thus is resistant to counter-persuasion and shows more word of mouth (WOM) behaviors). The proposed model examines the relationships between satisfaction and these two types of loyalty in the presence of mediating (switching cost and habit) and moderating (ideology and graphical design attractiveness) variables. The constructs in the model were selected following a review of the relevant literature from FLOSS and consumer behavior research. The IT artifact used to test the model is Firefox, and it was chosen for the following reasons. First, it is FLOSS that is known to be the most powerful driving force behind the increasing competition in the IS industry. Second, there exist other alternative web browsers in the market (e.g., IE, Safari, Opera, Flock) that provide similar features and user-interfaces, which lowers switching costs for users and thus makes loyalty more critical. Third, Firefox has a strong base of rigid loyal users who voluntarily dedicate their time and effort. The potential research contributions of the proposed study are two-fold. First, the study conceptualizes loyalty for IS post-adoption research in general, and for FLOSS in particular, and second, it attempts to empirically validate the role of loyalty in understanding continued IS use and the benefits generated by rigid loyal users.



# Impacts of Requirements Engineering Distribution in Open Source Software Development Projects

Veeresh Thummadi, Sean Hansen, and and Kalle Lyytinen

Case Westen Reserve University

**Abstract.** The wide scale geographic, social and temporal distribution in open source software development (OSSD) projects has made the requirements engineering (RE) process complex. We posit that these complexities can be addressed by understanding the impacts of structural distribution and governance on RE projects and outcomes in OSSD projects. In this study we seek to analyze different dimensions of RE distribution like spatial, structural, temporal, social and develop a framework for understanding the impacts of RE distribution on RE tasks in OSSD. This framework can show insights for improving the RE process in OSSD projects. We will investigate this using different OSSD projects like Mozilla, Apache, etc.



# Sustaining Processes within the Drupal Open Source Community

Sunah Suh

Graduate School of Library and Information Sciences, University of Illinois  
501 E. Daniel St, Champaign, IL 61820 sunahsuh@illinois.edu

**Abstract.** The Drupal started in 2000 as the software driving a community site for Dries Buytaert and a handful of friends. Since then, its popularity as an open-source content management system has exploded, finding its way to the backend of [whitehouse.gov](http://whitehouse.gov) and even the OSS conference website. The architecture of Drupal is strongly module-driven, with the core Drupal distribution only providing a fraction of the functionality available in the Drupal universe. As of August 2009, over 4600 free user-contributed modules were available on the [drupal.org](http://drupal.org) website. This highly-distributed development model piques questions about the social processes that keep the project from falling to pieces. Using Tamotsu Shibutani's framework of social processes, this project will examine the sustaining processes in place within this sociotechnical system. The data for this paper will largely come from forums, mailing list archives and documentation available on the [drupal.org](http://drupal.org) website. I will examine the text for patterns using grounded theory through the lens of Shibutani's theories. Particularly interesting are pages that self-describe the community and its processes which appear to contradict some of the more "on the ground" evidence in forums and mailing list discussions. A cursory examination of some of the available materials has yielded some interesting cases. These include: Many of the pages introducing Drupal convey the sense that this is a volunteer-driven community yet the most prolific developers and those in key decision-making positions nearly all make a living off of the Drupal project. How does this dichotomy contribute to the sustenance of this project? In the end, Dries Buytaert is the final decision maker on the Drupal core code—a benevolent dictatorship of sorts. This is an interesting juxtaposition to the democratic ideals of open-source software, yet this may be necessary for the arbiting dissention and maintaining coherence in distributed development. In terms of social stratification, the Drupal "Contribute" page lists many ways for non-developers to contribute to the project yet there's a clear power differential between developers and non-developers, made clear by documentation pages that speak of techniques for attracting developers to your issue and the economic incentives offered to developers as bounties that are not offered for other types of contributions such as documentation work or theming.

