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[1], 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.

[1] 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).

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 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 MacCabe's Cyclomatic Complexity and Halstead's Effort[2], 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

[2] 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.

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[3] 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

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

[3] measured as a function of the number of downloads, web traffic and development activity

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[4]. 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

[4] These rewrites were described in their wiki pages, respectively `http://live.gnome.org/EyeOfGnome/EogNg` and `http://live.gnome.org/SessionManagement/NewGnomeSession`.

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 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.

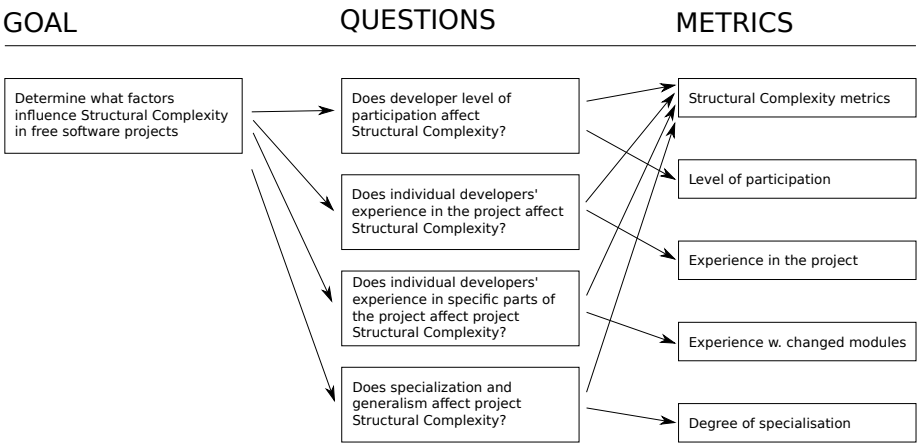

Fig. 1. GQM model of the research.

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. 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* — ΔSC. 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 ΔSC. When restricting the analysis to only those changes with positive or negative ΔSC, 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 a 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`[5] 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).
- 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.

[5] `http://git-scm.org/`. `git` has support for importing repositories from CVS and Subversion.

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, Chlara 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. Sftware 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 Héla Masmoudi. Channeling Firefox Developers: Mom and Dad Aren't Happy. In Barbara Russo, Ernesto Damiani, Scott A. Hissam, Björn 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. Héla 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, 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 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 Peripherial 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.