

On the effects of Refactoring in the Coordination of Software Development Activities

Cleidson R. B. de Souza, Maryanne P. Rosa, Crys S. Goto, Jean M. R. Costa, Pedro J. F. Treccani

Faculdade de Computação, Universidade Federal do Pará,
Belém, PA, Brasil

cdesouza@ufpa.br

Abstract. *Several empirical studies suggest that an alignment between the architecture of a software system and the coordination of development activities lead to better quality and improved performance. In this paper we investigate the possible effects of misalignments due to changes in the software architecture by describing the results of an exploratory study about the effects of refactoring in the coordination of software development activities in an open source project. We studied refactorings because they are perfect examples of changes in the software architecture. The project evaluated is the Jackrabbit, an Apache Software Foundation project. This project was analyzed using statistical tests and social networks analysis metrics. We evaluate different hypothesis regarding the impact of the refactoring process on project coordination. Initial results suggest that core software developers are especially affected by refactoring activities.*

Introduction

More than forty years ago, Conway (1968) suggested that the relationship between the architecture of a software system and the structure of the organization developing this software is homomorphic, the Conway's Law. Parnas (1972) similarly suggested a mechanism, the information hiding principle, to structure the software architecture in order to reduce software developers' coordination needs. Recently, these theoretical proposals have been corroborated by several qualitative (Staudenmayer 1997; Grinter 1999; de Souza et al. 2004; de Souza and Redmiles 2008) and quantitative (Sosa et al.

2004; Cataldo et al. 2006; Cataldo 2007) empirical studies. In sum, both researchers and practitioners recognize that the communication and coordination effort necessary to develop a software system is closely linked to its architecture. In fact, researchers have studied the alignment between the architecture and the coordination. For instance, software developers' performance is related to how well software developers align their coordination efforts with the dependencies in the software architecture, both at the team level (Staudenmayer 1997), and at the individual level (Cataldo 2007). On the other hand, misalignment between these aspects is seen as a possible explanation for breakdowns in software development projects (Brooks 1974; Bass et al. 2007). However, these studies are limited because they do not isolate the consequences of this misalignment: there are confounding factors.

In this paper we aim to investigate, in isolation, the possible effects of the misalignment between the software architecture and the coordination. Our approach in this paper is to study the effects of changes in the software architecture into software developers' coordination, since, in order to be successful when one aspect changes, the other must change accordingly (Staudenmayer 1997; Cataldo et al. 2006). We adopt a simple and informal definition of coordination based on the analysis of the work performed by software developers in their mailing lists. We report our results based on an analysis of a case study of software refactoring in an open source project. Software refactoring is the process of changing a software system in such a way that does not alter the external behavior, yet improves its internal structure (Fowler et al. 1999), therefore, it adequately represents changes in the software architecture. The coordination of the software development work is analyzed using statistical tests and social network analysis methods (Wasserman and Faust 1994). Our initial results suggest that key project members are affected by the coordination, since they perform additional work during the refactorings.

The rest of this paper is organized as follows. Our methods are described in the following section. Then, our results and discussion about them are presented. After that, we present our conclusions and future work.

Methodology

The research method used in this paper is an exploratory, single-case study (Yin, 2004). In any given case-study, it is necessary to detail how the case was selected. This is detailed in the following section.

Case Selection

In order to understand the effects of refactoring, we decided to analyze an open source project, i.e., a case is a software project. The project used is the Jackrabbit¹, an Apache

¹ <http://jackrabbit.apache.org/>

Software Foundation project. This project was selected because its information is publicly available in the internet, and, more importantly, it has had several refactorings since its inception.

This project has four different mailing lists: announcements (Jackrabbit Announce List), users (Jackrabbit Users List), development (Jackrabbit Development List), and source control (Jackrabbit Source Control List). In this work, we focused solely on the development list since we are interested in the coordination of *software developers*, the announcement list is used to inform users about new releases, and the source control list contains messages automatically generated by the configuration management tool.

Once our project was selected, we proceeded to collect and analyze its data as described in the following section.

Data Collection and Analysis

In order to collect data about the jackrabbit project we used a web-crawler named OSSNetwork (Balieiro et al. 2008) to extract information from the jackrabbit web-site. This tool allows one to: retrieve information from FLOSS repositories, store this information in a database, generate different types of social networks from this information, and, finally, analyze these networks.

Using OSSNetwork, we collected data about the development mailing list and stored it on a relational database. Queries to the database were performed to extract information like: thread duration (in days), total number of messages per thread, and number of different developers involved in each thread.

Data was analyzed in two ways: using statistical tests (Wild and Seber 1999) and using social network analysis methods (Wasserman and Faust 1994). Social networks were created based on the information extracted from the mailing list: software developers were the nodes of the network while edges were created when a developer, let's say B, replied to another developer, e.g. A. In this case, an edge from B to A is created (Wasserman and Faust 1994).

Identification of the Refactorings

We identified the refactorings in the project by reading messages in the mailing list and by reading the description within the bug-tracking system used, JIRA. We basically searched for the word “refactoring” in the mailing list and bug-tracking system. In the jackrabbit project, each refactoring is mapped to a JIRA issue so that there is a HTML page for each issue / refactoring containing information about the beginning and end date of the refactoring, the files modified, its author, among other types of information.

We identified a total of eight (8) refactorings in the project. Some of them happened in a short period of time (e.g., one day). We decided not to analyze these refactorings because we wanted to analyze the effects of refactoring while it was taking place (more details in the next section). In this paper we report the preliminary results from our analysis of two refactorings:

- Refactoring 1 (R1) – it took place between March 03, 2005 and March 16, 2005. This refactoring can be mapped to JCR-53 and JCR-66 in the JIRA tool; and
- Refactoring 2 (R2) – it took place between Jan 25, 2007 and March 07, 2006 and is mapped to issue JCR-309.

Once we defined the refactorings to be analyzed, it was necessary to define the periods of analysis for each refactoring.

Periods of Analysis

Refactorings R1 and R2 were analyzed according to two periods: *before* and *during* the refactorings. *Before* a refactoring means the period of one month before the refactoring started, while *during* a refactoring means the period in which the refactoring took place (start and end date of the refactoring according to the JIRA system). The period before refactoring R1 (R2) is labeled B1 (B2), while the period during refactoring R1 (R2) is called D1 (D2).

For each period of each refactoring, we calculated the mean and the standard deviation for each thread in the mailing list, the number of messages per thread and the total number of different developers who posted messages in the thread. If a developer posted more than one message in the same thread, he is counted as only one developer, but all his messages are taken into account.

Tests performed

We initially performed statistical tests to verify whether the data collected – thread duration, number of messages per thread and number of collaborators per thread – was normally distributed. Since this was not the case, we used non-parametrical tests (Wild and Seber 1999).

In our first test, we looked at thread duration, number of messages and number of collaborators per thread using the Mann-Whitney U test (Wild and Seber 1999). This test aims to compare independent samples for assessing whether two samples come from the same distribution. We compared periods B1 and D1, and B2 and D2 to find out information about these periods. Therefore, the Mann-Whitney U test will compare whether the duration of the threads before (B1) a refactoring is different from the duration of the threads during (D1) the same refactorings.

The second test aimed to compare the number of messages sent by each developer before and during each refactoring. In this case, we used the Wilcoxon signed-rank test, a non-parametric statistical test for two related samples or repeated measurements on a single sample (Wild and Seber 1999). Samples are related in this case because they describe the number of messages sent by the same developer before and during a refactoring. In this case, this test will evaluate whether the same developer sends a similar number of messages before and during a refactoring.

Finally, we tested the number of messages sent by a specific group of developers: the developers who were in the core (Borgatti 2000) of (i) the social network of the period before the refactoring, *and* of (ii) the social network created for the period during the refactoring. This test was performed only for those developers who were present in both social networks and who were in the core of each network to identify the impact caused by the refactoring in the most important developers of the project. Network core was identified using UCINET², a traditional social network analysis tool.

Results

As mentioned in the previous section, in our first test we looked at thread duration, number of messages and number of collaborators per thread. In this case, we found no statistical difference between periods before and during refactoring for refactoring R1 ($p=0,647$ for duration, $p=0,786$ for the number of messages and $p=0,636$ for the number of developers per thread), but a significant value for refactoring R2 ($p=0,004$ for duration, $p=0,016$ for the number of messages and $p=0,006$ for the number of developers per thread).

We tried to find a possible explanation for this result – a significant value in R2, but no significance in R1 – based on a qualitative analysis of these refactorings. We observed that refactoring R2 caused API changes across several modules, i.e., changes with broader impact than the other refactoring (R1), which changes a single module. To be more precise, the exact description of refactoring R2 extracted from the JIRA bug-system is:

“To better document and track the public JCR extensions and component API provided by Jackrabbit and to allow more room for refactoring within the Jackrabbit core, we should move (or create) the supported API interfaces to a new org.apache.jackrabbit.api package.”

In fact, by carefully examining the JCRs, we identified that R1 caused 110 changes in the source code, while refactoring R2 lead to 420 changes.

We also tested the average number of messages sent by each developer who posted messages in the mailing list during the analyzed periods. We found out that developers, during the periods of refactoring, send more messages than in the periods that precede a refactoring. This is true for both refactorings R1 and R2. To be more precise, we found some evidence of this result in R1 ($p=0,055$) and strong evidence in R2 ($p=0,0001$). This is an interesting result because it suggests that software developers are performing additional work during the refactorings.

However, this might not be a problematic situation per se because those performing additional work might be less important developers doing work in the periphery of the project. Therefore, we decided to perform the same test, but now considering only those

² <http://www.analytictech.com/ucinet/ucinet.htm>

developers who were in the core of the project, the key project members. In order to do that, we used social network analysis to identify the developers in the core and in the periphery of the project in both periods (before and during a refactoring) and for both refactorings (R1 and R2). We then used the same statistical test as before (Wilcoxon signed-rank) to evaluate the number of messages sent per developer, but now considering only those software developers who were in the core of the social network in both periods. We found out that developers in the core of the projects also send more messages during a refactoring. However, now we have very strong evidence supporting this result in both refactorings (R1 with $p=0,002$ and R2 with $p=0,004$).

Discussion

Our results suggest that the coordination of software development projects is in fact affected by refactorings of the source code. We initially evaluated discussion threads in the jackrabbit development mailing list and observed they were not affected, regarding its number of messages, duration or number of software developers involved for the refactorings analyzed, by the refactorings. This seems to suggest that the project community as a whole does not performed differently during the refactoring.

However, the results of our preliminary analysis indicate that some software developers send much more messages during the refactoring than in the period preceding it. More importantly, those software developers are the key members of the project, those who belong to the core of the social network created from the mailing list: they are the ones who are more active in the list. In general, we argue that we were able to find out novel and important effects of the misalignment of the software architecture and the coordination.

It is interesting to note two aspects: the overall number of messages per thread does not increase during refactoring, although the number of messages per contributor does change. We investigated this counter-intuitive result and observed that some contributors who *did not* participate in the discussions before the refactoring, *did* participate in the discussion during the refactoring. A possible explanation is that those developers were aware of the refactoring and decided to “jump in” the discussions to contribute. Additional research is still necessary to investigate alternative explanations.

In general, a consequence of our results is the recognition that changes in the software architecture need to carefully planned to not disturb even further core developers of the project. For instance, before a refactoring takes place, one needs to take into account its possible implications: core developers will need to prepare themselves to potentially guide other developers during the process. In addition, changes in the code need to be designed in order to avoid additional coordination problems. In fact, after performing the statistical tests, we analyzed the messages being exchanged during the refactorings. We were able to identify, at least, three situations in which software developers performed conflicting changes in the code, which lead to additional communication and coordination efforts.

Conclusions and Future Work

In this paper we describe our research on the effects of refactoring in the coordination of software development activities. This research is motivated by theoretical and empirical studies (Conway 1968; Parnas 1972; de Souza et al. 2004; Cataldo et al. 2006) regarding the socio-technical relationship between software architecture and coordination of development activities. Based on this socio-technical relationship, the hypothesis of the presented work is that refactoring, as it modifies the structure of the architecture, has effects in the coordination of software development activities. In this paper, we test this hypothesis with an exploratory single-case study, whose goal is to examine these effects in an open source project.

We analyzed the development mailing list of an open source project called jackrabbit using non-parametrical statistical tests and social network analysis and found out that while a refactoring is being performed, software developers who are central to the project engage in *additional* communication with their colleagues. We are not able to classify this effect as either negative or positive, but these results suggest the need to carefully plan refactorings in order to avoid an overload to core software developers in the project. It is important to emphasize that this paper reports on a single-case study in an open source project, therefore further research is necessary to generalize the results.

As for future work, we plan to analyze additional refactorings and projects in order to better understand the socio-technical relationship between software architecture and coordination, and its effects. We also plan to use qualitative methods to analyze the content of the messages exchanged during the refactorings.

Acknowledgments

This research was supported by the Brazilian Government under grants CNPq 479206/2006-6, CNPq 473220/2008-3 and by the Fundação de Amparo à Pesquisa do Estado do Pará (FAPESPA) through “Edital Universal N.º 003/2008”.

References

- Balieiro, M. A., et al. (2008). Facilitating Social Network Studies of FLOSS using the OSSNetwork. *International Conference on Open Source Systems*. Milan, Italy, Springer Series in Computer Science: 343-350.
- Bass, M., et al. (2007). Architectural Misalignment: An Experience Report. *IEEE/IFIP Working Conference on Software Architecture*.
- Borgatti, S. (2000). "Models of Core-Periphery Structures." *Journal of Social Network Analysis*. **21**: 375-395.
- Brooks, F. P. (1974). *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley.
- Cataldo, M. (2007). *Dependencies in Geographically Distributed Software Development: Overcoming the Limits of Modularity*. School of Computer Science. Pittsburgh, PA, Carnegie Mellon University. Ph.D.: 188.

- Cataldo, M., et al. (2006). Identification of Coordination Requirements: implications for the Design of Collaboration and Awareness Tools. *20th Conference on Computer Supported Cooperative Work*. Banff, Alberta, Canada, ACM Press.
- Conway, M. E. (1968). "How Do Committees invent?" *Datamation* **14**(4): 28-31.
- de Souza, C. R. B. and D. Redmiles (2008). An Empirical Study of Software Developers' Management of Dependencies and Changes. *International Conference on Software Engineering*. Leipzig, Germany, IEEE Press.
- de Souza, C. R. B., et al. (2004). How a Good Software Practice thwarts Collaboration - The Multiple roles of APIs in Software Development. *Foundations of Software Engineering*, Newport Beach, CA, USA, ACM Press.
- Fowler, M., et al. (1999). *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional.
- Grinter, R. E. (1999). System Architecture: Product Designing and Social Engineering. *Work Activities Coordination and Collaboration*, San Francisco, CA, USA, ACM Press.
- Parnas, D. L. (1972). "On the Criteria to be Used in Decomposing Systems into Modules." *Communications of the ACM*. **15**(12): 1053-1058.
- Sosa, M. E., et al. (2004). The Misalignment of Product Architecture and Organizational Structure in Complex Product Development. *Management Science*. **50**(12): 1674-1689.
- Staudenmayer, N. A. (1997). Managing Multiple Interdependencies in Large Scale Software Development Projects. *Sloan School of Management*. Cambridge, MA, USA, Massachusetts Institute of Technology. Ph. D.
- Yin, R. (2004). *Case study research: Design and methods*. Beverly Hills, CA: Sage Publishing.
- Wasserman, S. and K. Faust (1994). *Social Network Analysis: Methods and Applications*. Cambridge, UK, Cambridge University Press.
- Wild, C. J. and G. A. F. Seber (1999). *Chance Encounters: A First Course in Data Analysis and Inference*, John Wiley & Sons.