Analyzing Structural Software Changes: A Case Study

Črt Gerlec
Institute of Informatics
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova ulica 17, 2000 Maribor, Slovenia
crt.gerlec@uni-mb.si

Marjan Heričko
Institute of Informatics
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova ulica 17, 2000 Maribor, Slovenia
marjan.hericko@uni-mb.si

ABSTRACT
Software engineers strive to understand software's evolution and make software better and more reliable. In the last decade, software's life cycle became an interesting research area. However, software evolution could be analyzed from different aspects. In the study, we focused on analyzing structural software changes between software's versions. We used the WatiN project and observed its structural changes during the. The research showed that the number of structural source code changes increased near the release dates.

Keywords
Structural source code changes, code churn, software evolution

1. INTRODUCTION
A solid understanding of software development process allows engineers to develop software systems with a fewer bugs and better quality. Engineers also strive to produce a product with a high degree of reliability and maintainability. In order to achieve these goals, they have to understand a software development process and have a deeper insight in software structural evolution between releases or even versions.

Software development can be presented as a sequence of changes[1]. These changes are constant activities that add new functionalities to software, adapt it in order to fulfill new business demands, remove features that are not useful anymore and improve its internal structure for better maintenance. Usually, today's software systems are developed by more developers or even more development teams divided across countries. However, each developer has its own way of software development and uses own practices and patterns. All these facts impact on a software evolvement during a project development life cycle.

Software repositories are widely accepted software versioning and revision control systems in software engineering environment. They track changes that were done to a documents and a source code of a software system. Each change, which was done by a developer, is stored into a central repository. With other words, repositories contain a lot of information[2][3] and represent an archive of development facts. Therefore, such repositories allow researchers to analyze software evolution and reveal how a software systems and their structure are changing over time.

The motivation driving this study is to analyze structural software evolution between versions. Our aim is to check in what extent is software's structure changing over time. Especially, we will observe changes made close to the software's releases. The research could reveal some development patterns that are made before and after the software's releases.

The research is structured as follows. The section 2 presents a related work. Then, an essential background of the study is introduced in the chapter 3. The section 4 describes the research process and presents the results. In the last section we conclude our findings and describe a future work.

2. RELATED WORK
Software change evolution is an interesting research topic. Therefore, we can find various analyses regarding the evolution of software changes. Fluri and Gall[5] have developed an approach for analyzing and classifying change types based on code revisions. The approach differentiates between several types of changes on the method or class level. In the study, conducted by Confora et al.[6], a technique for identifying a CVS changes is presented. A technique detects modified lines where non-relevant added or deleted lines are excluded. For identifying line changes they used the Levenshtein edit distance algorithm and presented the technique in a case study. Research in [7] presented an approach for tracking a source code change evolution based on an algorithm that overcomes the Unix diff's versioning limitation. It is oriented towards software syntax and entity modifications.

Hall and Munson have presented an idea how to assess the amount of change in the complexity of the system across successive software builds[4]. The idea resulted in defining a code delta and a code churn. E. Ginger et al.[8] compared fine-grained source code changes and code churn in order to predict bugs in software system. They analyzed source code changes and used machine learning algorithms to empirically evaluate the performance of the approaches. The research, conducted by Nagappan and Ball[9], presented a set of relative code churn measures that relate the amount of churn to other variables like component size or temporal extent of churn. Finally, they compared absolute and relative measures together. The study in [10] also used code churn approach. Authors used object-oriented code metrics, xml code metrics and organizational metrics to predict yearly cumulative code churn of software projects. Results showed that code metrics and xml metrics are complementary to organizational metrics in order to estimate code churn. The research in [11] presented a change burst. The term represents a code fragments that are continuously changed over some period of time. It is defined with a gap size and a burst size. Arbuckle[12] presented an approach for measuring evolution of a multi-language software system. He avoids difficulties related to syntax, semantics and language paradigms by looking directly at relative shared information content. The approach measures a

BCI'12, September 16–20, 2012, Novi Sad, Serbia.

Local Proceedings also appeared in ISBN 978-86-7031-200-5, Faculty of Sciences, University of Novi Sad.

relative number of bits of shared binary information between artifacts of consecutive releases.

3. ANALYZING SOFTWARE CHANGES

3.1 Measuring Code Churn

The field of software change analysis has become very interesting research area in the last years. First analysis used single software's snapshot to evaluate software quality. Today, studies are focused on a whole software development life cycle where software repositories are analyzed. The important study was conducted by Hall and Munson[4]. They defined the code churn approach. Today, it is widely used in the field of evaluating software evolution and it is defined as follows:

$$\nabla_{ABC}^{j,j+1} = \sum_{c \in M_c} \left| m_c^{j+1} - m_c^j \right| + \sum_{a \in M_a} m_a + \sum_{b \in M_b} m_b,$$

where:

- ABC – represents a metric,
- M_c - represents a set of modules in both version,
- M_a - represents set of added modules,
- M_b - represents set of removed modules,
- m_a and m_b - represents added and removed module and
- m_c^j - represents the value of ABC metric for module *c* in the *j*th version.

Several studies used code churn and made some modifications to the definition above. For example, the code churn, that basis on lines of code, is frequently calculated just with a summation of added and deleted lines.

3.2 Measuring Structural Source Code Changes

The goal of our study was to analyze structural software changes during software's evolution. We defined a structural change as a change that transforms an object-oriented element (e.g. class, method, field).

We developed a tool that extracts software's versions from software repositories. Then, a special mechanism is used in order to detect structural software changes from successive versions. It uses different rules that are applied on a source code. The tool and its process of detecting structural software changes are presented in [14].

The tool for identifying source code changes supports several change types described in [13]. The supported change types are:

- add parameter, field and method,
- remove parameter, field and method,
- hide and unhide method,
- rename method,
- move attribute, method and class,
- extract superclass, interface and class
- pull up field and method,
- push down field and method and
- inline class.

In order to cover as many change types as possible, we added additional types to the list above. Additional types are add property, remove property, move property, pull up property, push down property and method body change. We added property changes in order to support the C# programming language. However, the language has a special object-oriented construct for properties. On the other hand, the Java programming language supports properties by defining an attribute and corresponding *getter* and *setter* methods.

4. THE RESEARCH

In the research, we analyzed WatiN project[15]. It is an open-source toolkit that is used to automate browser-based tests during software development. However, the toolkit is in development for more than 6 years and therefore contains lot of information in its software repository.

We extracted 1216 revisions from software's repository and analyzed them. We identified 587 source code files that were changed during the evolution. The source code was committed into the repository by 6 different developers and our change detection process identified 10898 structural source code changes. The table 1 shows the project's properties.

Table 1: The WatiN Project Properties.

Property	Value
Project start date	29.4.2006
Number of revisions	1216
Number of developers	6
Number of changed code files	587
Number of detected changes	10898

We grouped structural code changes in 5 different groups. Each group measures an extent of a code change. In the first three groups we used simplified code churn approach in order to evaluate a change extent of fields, properties and methods. We calculated them as follows:

$$Field\ Churn = Added\ Fields + Deleted\ Fields$$

$$Property\ Churn = Added\ Properties + Deleted\ Properties$$

$$Method\ Churn = Added\ Methods + Deleted\ Methods + Changed\ Methods$$

In the other two groups we just count the changes that moved different object-oriented constructs (i.e. move attribute, move method, move class) and count changes that were applied beyond the classes (i.e. extract superclass, extract interface, extract class, pull up field, property and method and push down field, property and method).

The figure 1 and figure 2 show the extent of changes made during the project's evolution. The latter figure shows a method changes during the evolution. On the other hand, the figure 1 shows other changes (e.g. field, property changes). The red dots represent the software's releases. The analysis showed that the method churn is much higher than other churns and changes. However, such an evolution was expected.

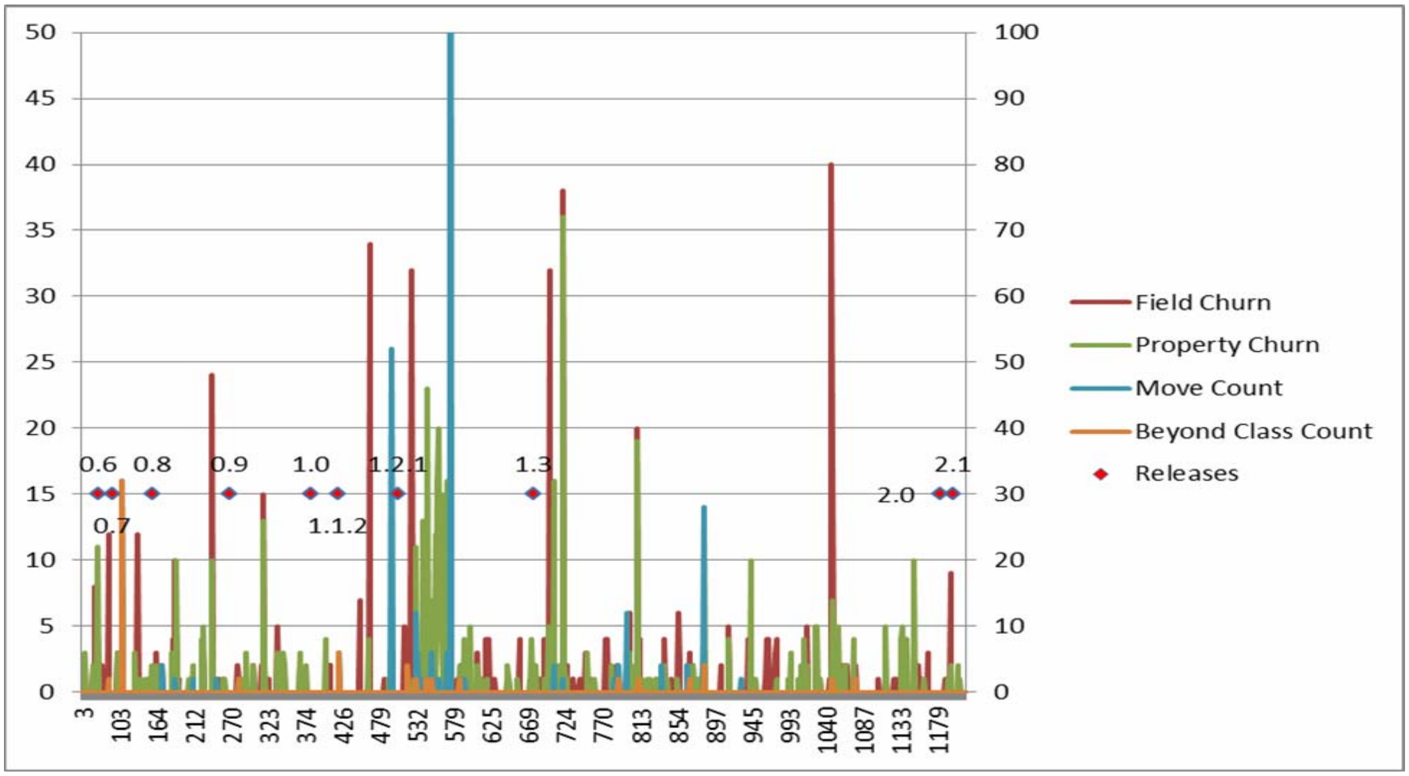

Figure 1: Fields, properties, move and beyond class changes during the project's evolution.

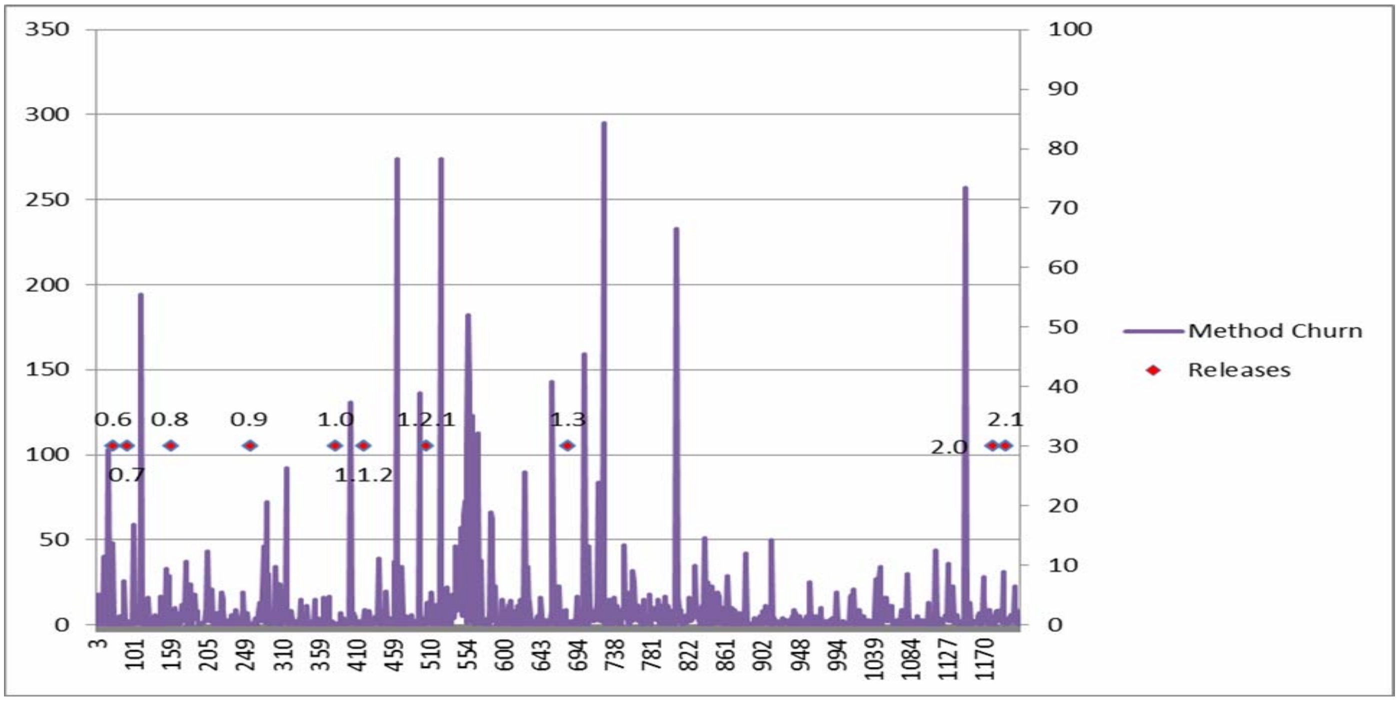

Figure 2: A method changes during the project's evolution.

The study also showed that in the first three releases (i.e. 0.6, 0.7, 0.8) there were lot of different type of changes. Beside field, property and method churn there were also changes that act beyond the classes. The latter changes could indicate on a restructuring in the software. The next interesting note is the high field churn before the release 0.9. Furthermore, the most changes were done between the releases 1.0 and 1.3. Beside changes in fields, properties and methods, there were also the move changes. We can assume that the release 1.3 was an important milestone in the project's evolution. Before the last two releases (i.e. 2.0 and 2.1), the change activity become more balanced. There are just three cases where the change activity (i.e. field churn, property and method churn) was high.

The figures also show that the change activity increased near the milestones (before and after). Such a pattern is recognized for all releases in the WatiN project. In general, there were a lot of changes in fields, properties and methods. On the other hand, we also detected some move and the "beyond class" changes. Such changes indicate that source code was refactored during the project's evolution.

5. CONCLUSION

In the research, the WatiN's structural evolution was analyzed. We observed the extent of changes made during the development life cycle and near the project's releases. We can conclude that structural change activity increased near the milestones. The highest change activity was detected between the versions 1.0 and 1.3. We assume that this release represents an important milestone in the project's evolution. After this release the change activity became more balanced. However, beside the basic changes (e.g. field, property and method changes), we also detected complex changes (e.g. beyond class and move changes) that indicate on a refactoring processes.

In the future work we would like to evaluate a correlation between a structural source code changes and a software quality (e.q. number of bugs) and use a machine learning algorithms to predicts software's quality.

6. REFERENCES

[1] Nagappan, N., Zeller A., Zimmermann T., Herzig K., Murphy B. 2010. Change Bursts as Defect Predictors. In Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE '10), 309-318, DOI=http://dx.doi.org/10.1109/ISSRE.2010.25.

[2] Kagdi H., Collard M. L., Maletic J. I. 2007. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J. Softw. Maint. Evol. 19, 2 (March 2007), 77-131. DOI=http://dx.doi.org/10.1002/smr.344.

[3] Fischer M., Pinzger M., Gall H. 2003. Populating a Release History Database from Version Control and Bug Tracking Systems. In Proceedings of the International Conference on Software Maintenance (ICSM '03). IEEE Computer Society, Washington, DC, USA.

[4] Hall G. A., Munson J. C. 2000.Software evolution: code delta and code churn, Journal of Systems and Software, Volume 54, Issue 2, Pages 111-118.

[5] Fluri, B.; Gall, H.C. 2006. Classifying Change Types for Qualifying Change Couplings. In Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC '06). IEEE Computer Society, Washington, DC, USA, 35-45. DOI=http://dx.doi.org/10.1109/ICPC.2006.16.

[6] Canfora G., Cerulo L., Di Penta M. 2007. Identifying Changed Source Code Lines from Version Repositories. In Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR '07). IEEE Computer Society, Washington, DC, USA, DOI=http://dx.doi.org/10.1109/MSR.2007.14.

[7] Canfora G, Cerulo L., Di Penta M.. 2009. Tracking Your Changes: A Language-Independent Approach. IEEE Softw. 26, 1 (January 2009), 50-57. DOI=http://dx.doi.org/10.1109/MS.2009.26.

[8] Giger E., Pinzger M., Gall H. C.. 2011. Comparing fine-grained source code changes and code churn for bug prediction. In Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11). ACM, New York, NY, USA, 83-92. DOI=http://doi.acm.org/10.1145/1985441.1985456.

[9] Nagappan N., Ball T. 2005. Use of relative code churn measures to predict system defect density. In Proceedings of the 27th international conference on Software engineering (ICSE '05). ACM, New York, NY, USA, 284-292. DOI=http://doi.acm.org/10.1145/1062455.1062514.

[10] Karus S. Dumas M. 2012. Code churn estimation using organisational and code metrics: An experimental comparison. Inf. Softw. Technol. 54, 2 (February 2012), 203-211. DOI=http://dx.doi.org/10.1016/j.infsof.2011.09.004.

[11] Nagappan N., Zeller A., Zimmermann T., Herzig K., Murphy B. 2010. Change Bursts as Defect Predictors. In Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE '10). IEEE Computer Society, Washington, DC, USA, 309-318. DOI=http://dx.doi.org/10.1109/ISSRE.2010.25.

[12] Arbuckle T. 2011. Measuring multi-language software evolution: a case study. In Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution (IWPSE-EVOL '11). ACM, New York, NY, USA, 91-95. DOI=http://doi.acm.org/10.1145/2024445.2024461.

[13] Fowler M, Kent B., Refactoring : improving the design of existing Code, Addison Wesley, 2002.

[14] Gerlec Č., Krajnc A., Heričko M., Božnik J., Mining source code changes from software repositories, Central & Eastern European Software Engineering Conference in Russia, 2012.

[15] The WatiN project, http://watin.org/.