Poster: Using Repository Data for Driving Software Architecture

Tobias Olsson
Department of Computer Science
Kalmar, Sweden
tobias.olsson@lnu.se

Morgan Ericsson
Department of Computer Science
Växjö, Sweden
morgan.ericsson@lnu.se

Anna Wingkvist
Department of Computer Science
Växjö, Sweden
anna.wingkvist@lnu.se

ABSTRACT

We are in the pursuit of establishing a method for continuous data driven software architecture. We describe the problem with current methods for measuring the impact of refactoring long lived systems at the architectural level and architecture compliance checking. We summarize our studies of code churn, productivity and an automatic tool for compliance checking. We conclude that architecture violations seem to play an important role, but current methods are infeasible for industry practice. Finally we propose to use repository data mining to improve current methods for architecture compliance checking.

CCS CONCEPTS

• **Software and its engineering → Software architectures**; *Software verification and validation*;

KEYWORDS

Open source system, technical debt, conformance checking, repository data mining

ACM Reference Format:
Tobias Olsson, Morgan Ericsson, and Anna Wingkvist. 2108. Poster: Using Repository Data for Driving Software Architecture. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3195094

1 INTRODUCTION

Long-lived software systems face the problem of continuous quality degradation if there is no effort to specifically prevent this [13]. These quality issues affect both the end product and the development process, e.g., performance problems, and degradation in engineering quality [6]. The root causes for this type of quality degradation are often architectural in nature and thus require maintenance at the architectural level [4, 12]; i.e., correcting problems in implementation related to architectural erosion or drift, or architecture reengineering to increase modularity or to accommodate for changes in the system or development environment, e.g., new frameworks or new ways to deliver and deploy the system.

The Technical Debt (TD) metaphor is used to describe the cost of degradation in quality, and more specifically Architectural Technical Debt (ATD) when applied to the architectural level [4]. Well managed TD can be used as a tool to monitor the trade-off between speed in development and accumulated quality degradation, i.e., controlled TD can be positive for the project as it ensures that features are delivered [4]. There are particular challenges to manage ATD that concerns how the debt should be measured when transitioning from one architecture to another or when enforcing architectural compliance. Generic architectural metric-based approaches are hard to adapt and their benefit is unclear [5, 7]. While some positive effects have been observed from using custom metrics, there is effort associated with the validation of such metrics and their measurement tools. There are also specific tools and methods to detecting architectural conformance problems. These methods typically need a specification of the desired architecture and a mapping of the implementation artifacts to the architecture. Based on these, the allowed dependencies according to the desired architecture are compared with the actual dependencies in the implementation [11]. Studies of such methods show that they are able to detect architectural divergences, but their impact and use in industry is low [1]. A major reason for this is the effort required to specify the desired architecture, map the implementation to architectural elements (which is a manual task). Another hindrance is that research and tools focus on using statically typed object-oriented implementation (e.g., Java), while large parts of the software industry have moved towards other paradigms (e.g. ECMAScript). These limitations are a serious hindrance for the use and benefit of such methods [1]. We think that architects should be able to freely explore new architectural ideas and determine the TD changes would impose to the system implementation. The system implementation as a whole should be continuously monitored for TD issues and improvements to either the implementation or architecture should automatically be suggested. This would allow architects and developers to make better decisions about refactorings, reuse, or abandonment in the face of architectural challenges. It would also allow simulation and evaluation of different architectural approaches. Repository data mining can give a broader insight in the evolution and development of a system. Incorporating analysis of code ownership, co-change of files, code churn, issue- and release management can give insights beyond static analysis of source code. We rely on such analysis to improve current methods for architectural conformance checking and investigate their cost, benefit, and usability.

In light of managing ATD, architecture conformance checking, and repository data mining, we derived on the following two research questions:

RQ1. What is the concrete impact of architectural violations and architectural refactoring?
RQ2. How can a non-homogeneous implementation be automatically mapped to a desired architecture?

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3195094

2 RESULTS

We have studied the evolution of architectural violations, technical debt, and code churn of the long-lived open source software JabRef [8, 9]. JabRef[1] is a Java desktop application to manage bibliographical references. The development started in 2003 without any documented or intended architecture. This resulted in a development crisis in 2015 when the developers decided to initiate an architectural refactoring effort to create a layered systems architecture, c.f. fig. 1. This new architecture was enforced by continuous architecture conformance checking implemented using unit tests.

We modeled productivity in the JabRef project by mining the project repository and analyzing developer activity and release logs. We also measured architectural violations and technical debt using two state-of-the-art tools, SonarCube[2] and JArchitect[3]. We found that we can define models that predict productivity based on such measures and that models based on TD measures predict productivity better if measures of architectural violations are included. This indicates that current tools, which based their analysis on low-level code quality indicators, can be enhanced by adding architectural conformance measures. We have also strengthened the hypothesis that architectural violations have an impact on developer productivity [8].

We also investigated the number of added and deleted lines of code in the largest files in versions before and after the start of the architectural refactoring. We found that code churn was significantly higher for files that contained architectural violations before the refactoring compared to similarly sized files (without violations) during the same period. This abnormal change pattern was not significant after the refactoring. We also investigated code ownership in these files but could not find any difference in code ownership between files with violations compared to normal files, neither before nor after the architectural refactoring [9].

We have implemented Rulzor, a fully automatic architectural conformance checking tool that integrates with the project repository [10]. When a developer commits a change to the repository (code), an automatic analysis is started. If any architectural violations are detected an issue that contains the offending file and possible cause of the violations is posted to the project repository. The developer can then fix the problem and the issue will be automatically resolved. We tested Rulzor on a group of students and found that there were a significantly lower amount of architectural violations in projects that used the tool compared to the control groups. This suggests that tools for continuous data-driven architecture are both feasible and beneficial [10].

3 CONCLUSION AND FUTURE WORK

We have observed a tangible impact of architectural violations on productivity and code churn in a project that dealt with an architectural refactoring situation **RQ1**. However, to perform architectural violation measures we have been limited to one system with a known implementation to architecture mapping. Our tools have used this mapping to automatically cluster classes in the implementation to the architecture. This is not a feasible approach for further studies or industry applications. To improve this issue, we turn our attention to **RQ2** and are investigating methods to perform semi-automatic clustering of non-homogeneous code bases. We will build on previous work related to architecture conformance [2, 3] and investigate the possibility to use repository data, e.g. co evolution of files, in such methods. We are working on a new tool (S4RDM3X) for software architecture repository data mining and exploration. That will enable us to study the evolution of more projects and provide industry with a starting point for a data driven approach to software architecture.

Figure 1: Productivity during the evolution of JabRef.

REFERENCES

[1] Nour Ali, Sean Baker, Ross O'Crowley, Sebastian Herold, and Jim Buckley. 2017. Architecture consistency: State of the practice, challenges and requirements. *Empirical Software Engineering* (2017), 1–35.

[2] Roberto Almeida Bittencourt, Gustavo Jansen de Souza Santos, Dalton Dario Serey Guerrero, and Gail C Murphy. 2010. Improving automated mapping in reflexion models using information retrieval techniques. In *Reverse Engineering (WCRE), 2010 17th Working Conf on.* IEEE, 163–172.

[3] Andreas Christl, Rainer Koschke, and M-A Storey. 2005. Equipping the reflexion method with automated clustering. In *Reverse Engineering, 12th Working Conf on.* IEEE, 10–pp.

[4] Neil Ernst, Stephany Bellomo, Ipek Ozkaya, Robert Nord, and Ian Gorton. 2015. Measure it? manage it? ignore it? software practitioners and technical debt. In *Proc of the 10th Joint Meeting on Foundations of Software Engineering.* ACM, 50–60.

[5] Heiko Koziolek. 2011. Sustainability evaluation of software architectures: a systematic review. In *Proc of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS.* ACM, 3–12.

[6] Antonio Martini, Jan Bosch, and Michel Chaudron. 2015. Investigating Architectural Technical Debt accumulation and refactoring over time: A multiple-case study. *Information and Software Technology* 67 (2015), 237–253.

[7] Robert Nord, Ipek Ozkaya, Raghvinder Sangwan, and Ronald Koontz. 2014. Architectural dependency analysis to understand rework costs for safety-critical systems. In *Companion Proc of the 36th Int Conf on Software Engineering.* ACM, 185–194.

[8] Tobias Olsson, Morgan Ericsson, and Anna Wingkvist. 2017. Motivation and Impact of Modeling Erosion Using Static Architecture Conformance Checking. In *Software Architecture Workshops (ICSAW), IEEE Int Conf on.* IEEE, 204–209.

[9] Tobias Olsson, Morgan Ericsson, and Anna Wingkvist. 2017. The Relationship of Code Churn and Architectural Violations in the Open Source Software JabRef. In *Proc of the 11th European Conf on Software Architecture: Companion Proc (ECSA '17).* 152–158.

[10] Tobias Olsson, Daniel Toll, Morgan Ericsson, and Anna Wingkvist. 2016. Evaluation of an architectural conformance checking software service. In *Proccedings of the 10th European Conf on Software Architecture Workshops.* ACM, 15.

[11] Leonardo Passos, Ricardo Terra, Marco Tulio Valente, Renato Diniz, and Nabor das Chagas Mendonca. 2010. Static architecture-conformance checking: An illustrative overview. *IEEE software* 27, 5 (2010), 82.

[12] Jilles Van Gurp and Jan Bosch. 2002. Design erosion: problems and causes. *Journal of systems and software* 61, 2 (2002), 105–119.

[13] Liguo Yu and Alok Mishra. 2013. An empirical study of Lehman's law on software quality evolution. *Int Journal of Software & Informatics* 7, 3 (2013), 469–481.

[1] http://jabref.org
[2] https://www.sonarqube.org
[3] https://www.jarchitect.com