Software Development Data for Architecture Analysis: Expectations, Reality, and Future Directions

Yuanfang Cai
Computer Science Department
Drexel University
Philadelphia, PA, USA
yfcai@cs.drexel.edu

Rick Kazman
Shidler College of Business
University of Hawaii
Honolulu, HI, USA
kazman@hawaii.edu

ABSTRACT

Recently we have worked with a dozen industrial collaborators to pinpoint and quantify architecture debts, from multi-national corporations to startup companies. Our technology leverages a wide range of project data, from source file dependencies to issue records, and we interacted with projects of various sizes and characteristics. Crossing the border between research and practice, we have observed significant gaps in terms of data availability and quality among projects of different kinds. Compared with successful open source projects, data from proprietary projects are rarely complete or well-organized. Consequently, not all projects can benefit from all the features and analyses we provide. This, in turn, made them realize they needed to improve their development processes. In this talk, we categorize the commonly observed differences between open source and proprietary project data, analyze the reasons for such differences, and propose suggestions to minimize the gaps, to facilitate advances to both software research and practice.

KEYWORDS

Software Architecture, Technical Debt, Architecture Debt, Software Modularity, Software Quality

1 Topic Description

The gap between software engineering research and practice has hindered progress on both fronts. High quality data is critical for software research, from bug prediction to architecture analyses to visionary knowledge graphs. Much software engineering research leverages data collected from open source projects, and it is rare that published research tools are applied in practice, and rarer still that they produce similarly impressive results.

The reasons behind this gap are complex. In the past few years, our architecture debt research has had the privilege of "crossing the border" and being applied in practice. Our industrial collaborators cover a wide spectrum from multi-national corporations to startups. During the process of working with them, we witnessed key differences between open source and proprietary projects in terms of how they are managed, how development data are collected, and how data quality differs. These differences have presented challenges for our techniques to be applied in practice to closed-source projects. But they have also opened up opportunities for us to improve our research and have motivated our collaborators to improve their development processes.

ICSE-SEIP '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7123-0/20/05.
https://doi.org/10.1145/3377813.3381357

In this talk, we first explain how we leverage heterogenous data sources generated from a development process to identify and quantify architecture debts, so that both development teams and management can share a common goal. This goal is to understand the severity and scope of their debts, to estimate the costs of refactoring them, and finally, to calculate and validate the benefits of refactoring. Our basic idea is to integrate structural information reflected in the source code with source files transactions, as recorded in the project's revision history, to calculate key performance indicators and to estimate costs and benefits.

Our tool suite consists of a novel maintainability score—*decoupling level (DL)* [4], a technique for detecting *architecture anti-patterns*—recently defined and extensively validated structural flaws [15], and a technique for detecting *architecture roots*—file groups responsible for high maintenance costs [6]. Next we briefly explain each of these concepts.

(1) *Decoupling Level* (DL) [4] measures how well a software system is *decoupled* into small and independent modules that can be developed and maintained in parallel. DL measures various types of dependencies among source code entities, both structural and evolutionary couplings.

(2) *Architecture anti-patterns* [3] are recurrent patterns of file relationships that violate widely accepted design principles and that have been shown to be strongly correlated with high maintenance costs. The detection of these anti-patterns requires a project's revision history that records how files were changed together. To quantify the penalty incurred by each anti-pattern instance, we measure the number of bugs and changes, as well as the bug churn and change churn, using project history data, such as a that available from, for example, a Git log.

(3) *Architecture roots* [5] are defined as architecturally connected file groups that capture a project's most error-prone or change prone files. We first proposed the notion of a *design rule space* (DRSpace)—a set of files implementing a pattern, a feature, or other important system concerns—and an architecture root is a type of DRSpace. Our study showed that five (or fewer) architecture roots in a project typically cover 50% to 90% of the project's most error-prone files. To calculate roots, we need three types of data: dependencies among source code entities, revision history, and issue records, with the assumption that each commit can be linked to an issue ticket.

Our tool suite performs these analyses and automatically generates a report summarizing the discovered architecture problems, and provides detailed data for follow-on analysis. We have multiple publications reporting case studies with industrial collaborators, e.g. [2], [11], [12], [13], [14]. Like most software research tools, before applying these techniques on industrial projects we first extensively evaluated them using open source projects. But we found that it is much harder to extract data from commercial projects: not all development teams have all the data available and the data are usually not of high quality. The teams did not realize the problems in their development processes until our tools reported unexpected results.

To discover the reasons behind these abnormal results, we intensively interacted with team members and observed several representative cases that illuminated the differences. In this talk, we categorize the differences between open source data and proprietary project data and analyze the reasons behind these gaps based on our extensive interactions with developers, designers, architects and managers. These experiences have motivated us to improve our research and motivated the industrial development teams to improve their development processes. Based on these experiences, we propose suggestions to minimize the gaps, to facilitate advances to both software research and practice.

2 Speaker Names and Bios

2. Speaker Names

Dr. Yuanfang Cai, Associate Professor, Drexel University

Dr. Rick Kazman, Professor, University of Hawaii

2. Bio of Yuanfang Cai

Yuanfang Cai is a Professor at Drexel University. Her major research interests include software architecture, design, modularity, evolution, the revelation of architecture and architectural issues within source code, and the integration of software architecture and economics. She has authored more than 80 publications and is ranked the 2nd in the field of Technical Debt and Software Modularity on Google Scholar. She is the creator of the suites of concepts and technologies leading to new ways of software architectural analysis: the design rule hierarchy algorithm to model software architectures [8], the notion of DRSpace that integrates software architecture modeling with evolution history and quality information [6], the Decoupling Level (DL) metric that measures to what extend a software architecture is decoupled [4], and the approaches to quantifying key parameters of architecture debts [7].

2. Bio of Rick Kazman

Rick Kazman is a Professor at the University of Hawaii and a Visiting Scientist at the Software Engineering Institute of Carnegie Mellon University. His primary research interests are software architecture, design and analysis tools, software visualization, and software engineering economics. Kazman has co-created several highly influential methods and tools for architecture analysis, including the SAAM (Software Architecture Analysis Method), the ATAM (Architecture Tradeoff Analysis Method), the CBAM (Cost-Benefit Analysis Method) and the Dali and Titan tools. He is the author of over 200 publications, and co-author of several books, including Software Architecture in Practice [10], Designing Software Architectures: A Practical Approach [9], Evaluating Software Architectures: Methods and Case Studies, and Ultra-Large-Scale Systems: The Software Challenge of the Future.

Both authors have co-authored most of the papers that are the foundation of this talk [1][2][3][4][5][6][7][11][12][13][14][15].

REFERENCES

[1] Y. Cai, R. Kazman, C., Jaspan, J. Aldrich: Introducing tool-supported Architecture review into software design education. CSEE&T 2013

[2] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka. A case study in locating the architectural roots of technical debt. In Proc. 37th International Conference on Software Engineering, May 2015.

[3] R. Mo, Y. Cai, R. Kazman, and L. Xiao. Hotspot patterns: The formal definition and automatic detection of architecture smells. In Proc. 12thWorking IEEE/IFIP International Conference on Software Architecture, May 2015.

[4] R. Mo, C. Yuanfang, K. Rick, X. Lu, and F. Qiong. Decoupling level: A new metric for architectural maintenance complexity. In Proc. 38th International Conference on Software Engineering, pp. 499–10, 2016.

[5] L. Xiao, Y. Cai, and R. Kazman. Titan: A toolset that connects software architecture with quality analysis. In 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering, 2014.

[6] L. Xiao, Y. Cai, and R. Kazman. Design rule spaces: A new form of architecture insight. In Proc. 36rd International Conference on Software Engineering, 2014.

[7] L. Xiao, Y. Cai, R. Kazman, R. Mo, and Q. Feng, "Identifying and quantifying Architectural Debts," in Proc. 38rd International Conference on Software Engineering, 2016.

[8] S. Wong, Y. Cai, G. Valetto, G. Simeonov, and K. Sethi. Design rule hierarchies and parallelism in software development tasks. In Proc. 24th International Conference on Automated Software Engineering, pp. 197–208, 2009.

[9] H. Cervantes, R. Kazman, *Designing Software Architectures: A Practical Approach*, Addison-Wesley, 2016.

[10] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, 3rd ed., Addison-Wesley, 2012 (2nd ed. 2003, 1st ed. 1998).

[11] R. W. Schwanke, L. Xiao, Y. Cai: Measuring architecture quality by structure plus history analysis. ICSE 2013: 891-900

[12] W. Wu, Y. Cai, R. Kazman, R. Mo, Z. Liu, R. Chen, Y. Ge, W. Liu, J. Zhang: Software Architecture Measurement - Experiences from a Multinational Company, in Proc. of the 12th European Conference on Software Architecture, (ECSA'18), pp 303-319, Sept, 2018.

[13] R. Mo, Y. Cai, R. Kazman, Q. Feng: Assessing an architecture's ability to support feature evolution, in Proc. of the 26th Conference on Program Comprehension (ICPC'18), pp 297-307, May 2018

[14] M. Nayebi, Y. Cai, R. Kazman, G. Ruhe, Q. Feng, C. Carlson, F. Chew: A longitudinal study of identifying and paying down architecture debt, in Proc. of the 41st International Conference on Software Engineering: Software Engineering in Practice ICSE'19 (SEIP), pp 171-180, May, 2019

[15] R. Mo, Y. Cai, R. Kazman, L. Xiao and Q. Feng: Architecture Anti-patterns: Automatically Detectable Violations of Design Principles, *IEEE Transactions on Software Engineering*, 2019.