Recovering design intent from refactorings

Thomas Schweizer
Department of Computer Science and Operations Research
Université de Montréal
Montréal, Canada
thomas.schweizer@umontreal.ca

***Abstract*—Understanding design decisions from the past is essential during development to ensure continuity in a system and maintain quality. However, such information is hard to retrieve for various reasons such as misleading, missing documentation or design erosion. We postulate that studying a project's refactoring history can help uncover design information and propose an initial exploratory approach to test our intuitions. We measure metric changes between revisions containing refactorings and explore if there is a correlation between the presence of design intents and metric changes. Our preliminary results show that our approach identifies revisions containing design decisions with 67% accuracy in average. We describe our methodology in detail and the next steps we are taking to explore this problem. The results indicate that solutions seem to exist to narrow down and target specific signature of changes in a revision.**

***Index Terms*—design recovery, tradeoffs, revision history, refactoring**

I. Introduction

While making changes to an existing system, developers need to understand the context in which they work. To this end, they need to read the code of each relevant artifact (methods, classes, modules) and very often also look at the history of in order to understand the intent of the role the artifact is supposed to have. Even when documentation is available, this task is time-consuming because the developer needs to browse the revision history. Our long term goal is to offer a tool that enables developers to quickly identify revisions that contain major design decisions made to a system. We see two significant incentives: 1) Improve the comprehension behind the processes involved in building a software application. 2) Increase the number of options developers have to navigate the development history of an artifact and identify its role of in a system.

In this paper, we present a preliminary approach to determine if we can identify interesting revisions likely to contain design decisions. The idea is to simplify the scope of research by looking at revisions containing refactorings; we refer to such revisions as "refactoring revisions" (RRs). Refactoring has a clear impact on the design of a software: developers purposefully use refactoring to express specific design intentions [1] and use recommender systems to identify the most suitable refactorings to best suit their intents [2]. In this context, we examined the development history and refactoring activity of 11 software projects. Our contributions are the following: We studied the fluctuations in metrics for their revisions, finding that refactoring activities often lead to tradeoffs in metrics. Then, we performed a qualitative analysis by manually annotating a sample of all refactoring commits, finding that, on average, 67% of refactoring commits involving tradeoffs also carry design intent. These observations allowed us to draw conclusions about the relationship between refactoring, tradeoffs and design intent. Notably, we find that fluctuations to a handful of internal quality metrics are already a good indicator to find commits in a projects version history that carry design intent. These are very encouraging preliminary results that lend confidence to our intuition to further research the topic.

In Section II we describe the framework of the study and in section III, we present the results. We discuss related work in Section IV and conclude in Section V.

II. Overview

a) Project selection: We analyzed 11 popular open-source Java projects. JFreeChart, JUnit4, Ant, jEdit, JMeter, and ArgoUML were selected for their usage in past academic studies and Retrofit, OkHttp, Dagger2, RxJava, and Jena to diversify our dataset. In addition, we based our selection on the projects' popularity, size, number of contributors, platform, type (e.g., library, desktop application), and development style diversity. The smallest project is Retrofit with 1591 revisions, and the largest project is ArgoUML with 17795 revisions. The median is 5314 revisions.

b) Pipeline: We built a pipeline capable of processing the revisions of a project and identify which ones contain a design decision. The pipeline is made of independent modules that can be replaced or customized easily. The process is the following: 1) We extract the RRs using RMiner [3]. 2) We compute a suite of metrics for all RRs and their parent revisions with our tool. The metrics are computed by SourceMeter [4] which is plugged in our tool. 3) Our tool aggregates the metrics for each revision and calculates the metric fluctuations between a revision and its parent. 4) An RScript classifies each revision in function of the pattern of its metric fluctuations. For our preliminary investigation, we adopt a naive approach: if, in a RR, some metrics improve while others deteriorate, we postulate that a developer is consciously making a design decision that results in a tradeoff between metrics. We also record other interesting patterns of metrics fluctuations for further analysis (see Section III below).

c) Oracle: In order to compare the predictions of our initial approach, we built a reference dataset: We manually

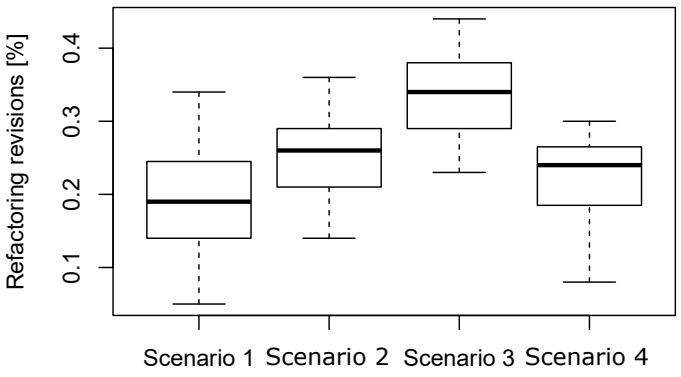

Fig. 1. Distribution of the refactoring revisions in each scenario across projects

annotated 186 unique RRs as containing a design decision or not. To calibrate the oracle, two reviewers independently analyzed the RRs found by RMiner in JFreeChart. The inter-rater agreement was moderate, indicated by a Cohens Kappa of 0.490. We refined our annotation protocol while resolving conflicts and then analyzed the remaining RRs separately. The remaining 106 represents the RRs extracted from the project Retrofit.

Additionally, we built another dataset by annotating a random sample of 10 revisions containing metric tradeoffs for each project (we reused the annotation decisions for JFreeChart and Retrofit). We plan to make this dataset public upon publication.

III. Results

A. Impact of refactoring on internal quality metrics

We calculated the metric fluctuations for each class for all the RRs of each project. Overall, we had 5921 RRs on which we calculated the metrics plus their respective parents. We defined 4 patterns (also called scenarios) of metric fluctuation: 1) No metric changes. 2) A change in a unique metric. 3) 2 metrics or more are improving or declining. 4) 2 metrics or more change in different directions. The last pattern is what we refer as a tradeoff in metrics. We found that scenarios are distributed more or less equally in RRs as suggested by figure 1.

B. Correlation between metric tradeoffs and design intent

In this part, we take all the RRs containing a tradeoff (scenario 4) and see if they actually contain a design intent using our oracle. In our sampled dataset, we find that in 67% of the cases, a RR belonging in scenario 4 also contains a design decision. For the full JFreeChart and Retrofit we obtain respectively 72.06% and 68.52%.

To understand these results in depth, we are currently looking at the following research questions: What metrics should be used characterized tradeoffs ? Which artifacts should be aggregated for a revision? What granularity of artifacts should be considered (e.g., package, class, method)?

IV. Related Work

Recovery of design decisions has been studied from an architectural perspective. Jansen et al. proposed a methodology for recovering architectural design decisions across releases of a software system [5]. The methodology provides a systematic procedure for keeping up-to-date the architecture documentation and prescribes the steps that the software architect must follow in order to identify and document architectural design decisions across releases. A fully automated technique for the recovery of architectural design decisions has been, recently, proposed by Shahbazian et al. [6]. The technique extracts architectural changes from the version history of a project and maps them to relevant issues from the project's issue tracker. Each disconnected subgraph of the resulting graph corresponds to an architectural decision. The recovered decisions are relevant to structural changes in system components, applied across successive releases. Our method focuses on decisions affecting detailed design and concern the structure of classes and the distribution of state and behavior among them. Moreover, decision recovery takes place at the revision level and is guided by metrics' fluctuations and fine-grained changes due to refactorings. Besides, we employ issue tracker information for manual cross-checking of design decisions as well as commit messages and source code comments.

The activity of refactoring and its relation to design has been extensively studied. Chavez et al [7] performed a large-scale study to understand how refactoring affects internal quality attributes on a microscopic level, or in other words on a metric basis. In contrast, our study aims at exploring the role of refactoring on a macroscopic level and how it relates to greater design decisions. Cedrim et al. [8] investigated the extent to which developers are successful at removing code smells while refactoring. Soetens et al. [9] analyzed the effects of refactorings on the code's complexity. Tsantalis et al. [10] investigated refactorings across three projects and examined the relationship between refactoring activity, test code and release time. They found that refactoring activity is increased before release and is mostly targeted at resolving code smells. Compared to the study presented here, that study was not guided by metrics, it did not involve extensive qualitative analysis and did not discuss more major design decisions as part of the intent.

V. Conclusion

The results suggest that metric tradeoffs in revisions containing refactoring are promising indicators to find design intent in revision histories. Our short term goal is to understand better the limits of identification by metrics of our naive approach.

Then, we will be able to evaluate and integrate additional heuristics such as comments, commit messages, code mining and the application of design principles such as SOLID. If this works well, we will be testing our approach outside of the refactoring context as well. In parallel, we want to expand our reference dataset and invite other researchers to use what we built so far.

REFERENCES

[1] D. Silva, N. Tsantalis, and M. T. Valente, "Why we refactor? confessions of github contributors," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 858–870. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950305

[2] G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto, *Recommending Refactoring Operations in Large Software Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 387–419. [Online]. Available: https://doi.org/10.1007/978-3-642-45135-5_15

[3] N. Tsantalis, M. Mansouri, L. Eshkevari, D. Mazinanian, and D. Dig, "Accurate and efficient refactoring detection in commit history," in *40th International Conference on Software Engineering (ICSE 2018)*. Gothenburg, Sweden: IEEE, May 27 - June 3 2018.

[4] R. Ferenc, L. Lang, I. Siket, T. Gyimthy, and T. Bakota, "Source meter sonar qube plug-in," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, Sept 2014, pp. 77–82.

[5] A. Jansen, J. Bosch, and P. Avgeriou, "Documenting after the fact: Recovering architectural design decisions," *Journal of Systems and Software*, vol. 81, no. 4, pp. 536 – 557, 2008, selected papers from the 10th Conference on Software Maintenance and Reengineering (CSMR 2006).

[6] A. Shahbazian, Y. K. Lee, D. Le, Y. Brun, and N. Medvidovic, "Recovering architectural design decisions," in *2018 IEEE International Conference on Software Architecture (ICSA)*, April 2018, pp. 95–9509.

[7] A. Chávez, I. Ferreira, E. Fernandes, D. Cedrim, and A. Garcia, "How does refactoring affect internal quality attributes?: A multi-project study," in *Proceedings of the 31st Brazilian Symposium on Software Engineering*, ser. SBES'17. New York, NY, USA: ACM, 2017, pp. 74–83. [Online]. Available: http://doi.acm.org/10.1145/3131151.3131171

[8] D. Cedrim, L. Sousa, A. Garcia, and R. Gheyi, "Does refactoring improve software structural quality? a longitudinal study of 25 projects," in *Proceedings of the 30th Brazilian Symposium on Software Engineering*, ser. SBES '16. New York, NY, USA: ACM, 2016, pp. 73–82. [Online]. Available: http://doi.acm.org/10.1145/2973839.2973848

[9] Q. D. Soetens and S. Demeyer, "Studying the effect of refactorings: a complexity metrics perspective," in *International Conference on the Quality of Information and Communications Technology*. IEEE, 2010, pp. 313–318.

[10] N. Tsantalis, V. Guana, E. Stroulia, and A. Hindle, "A multidimensional empirical study on refactoring activity," in *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, ser. CASCON '13. Riverton, NJ, USA: IBM Corp., 2013, pp. 132–146. [Online]. Available: http://dl.acm.org/citation.cfm?id=2555523.2555539