Quantifying the Effects of Refactorings on Bad Smells

Cleiton Tavares[1](Mastering), Mariza A. S. Bigonha[1](Advisor), Eduardo Figueiredo[1](Co-advisor)

[1] Master's degree in Computer Science
Graduate Progam in Computer Science
Department of Computer Science – Federal University of Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

Entry date: August 2018 - Proposal: May 2019 - Expected date: First Semester of 2021

{cleiton.silva,mariza,figueiredo}@dcc.ufmg.br

Abstract. *Refactoring aims to remove bad smells and increase software maintainability by improving the software structure without changing its behavior. Even with the existence of tools to assist refactoring, many developers do not trust in their solutions, claiming that some studies show that refactoring may even introduce new bad smells into the source code. However, we do not find a complete catalog that states when this may occur. To investigate this subject deeply, the goal of this dissertation is to evaluate the effects of refactoring on the detection of bad smells. Specifically, we want to know if and what refactoring removes bad smells or introduces them. To achieve our goal, we plan to conduct empirical studies to provide a catalog showing these situations.*

Keywords: *Bad Smell, Refactoring, Impacts of Refactoring*

Related CBSoft symposia: SBES

1. Introduction

Bad smell is a code structure that we may resolve through refactoring. The purpose of code refactoring is to remove bad smells in the code by modifying its internal structure to improve code maintenance [Fowler et al. 1999]. Therefore, we may establish a direct relationship between refactoring and bad smell.

Several studies show different strategies for the refactoring application. Some studies focus on quality metrics [Kataoka et al. 2002, Moser et al. 2006], others present opportunities for refactoring motivated by the detection of bad smells and anti-patterns [Boussaa et al. 2013, Kessentini et al. 2010, Moha et al. 2009], while others describe that refactoring may solve the bad smells present in the source code [Fokaefs et al. 2011, Tsantalis and Chatzigeorgiou 2009]. However, studies also show that the refactoring process often does not solve them [Bavota et al. 2015, Cedrim et al. 2017]. A possible strategy that may help make the refactoring process more effective in solving the identified bad smells is to consider the existence of their relationship. However, we must refactor in a disciplined way to not introduce new bad smells in the source code [Mkaouer et al. 2016, Pietrzak and Walter 2006]. As a direct example, Pietrzak and Walter (2006) found out that a *Move Method* applied to a *Lazy Class* may result in a new bad smell, *Feature Envy*. The fact that *Move Method* transfers the envious method may reduce the responsibility of a class.

A software system is in the process of constant change and evolution. These changes may become complex over time if there is no quality in the software system's structure. However, as noted, the number of bad smells removed effectively by the refactoring process is still deficient, and even worse than that, it may not only remove bad smells but also introduce them. This fact may indicate why developers are less and less concerned with solving problems in a software system's internal structure. Thus, it is essential to understand how bad smells may behave in the context of their identification and then take practical actions to maintain or improve the software system's structural quality.

This dissertation aims to address these problems by conducting empirical studies to investigate the effect of refactoring on the resolution of bad smells in software systems. Specifically, using some bad smells proposed by Fowler, we want to investigate if and what refactoring operations remove or introduce them. One of our most important contributions would be a catalog reporting the refactorings' effects on the bad smells chosen. Our preliminary results show that an exemplar refactoring tool, JDeodorant [Fokaefs et al. 2011], introduces many bad smells after applying automated refactoring.

2. Background and Related Work

Bad smell is evidence of problems in the code structure that may be resolved via refactoring. Fowler (1999) proposed one of the complete lists containing 22 bad smells; besides that, he describes how we may identify them and what refactoring strategies we may apply in their solution. *Feature Envy* is an example of a bad smell that occurs when a function in one module spends more time communicating with functions or data inside another module than it does within its module.

Refactoring is a process of improving the software system's internal structure without changing the code's external behavior. One of the most well-known and complete catalogs presents a list of 72 refactorings [Fowler et al. 1999]. *Extract Class* is an example of refactoring, which consists of splitting a class that is too big to understand with many methods and quite a lot of data to create a new class. We may perform this refactoring when (1) the data subset and the methods' subset seem to go together, and (2) some data subset usually change together or are mainly dependent on each other.

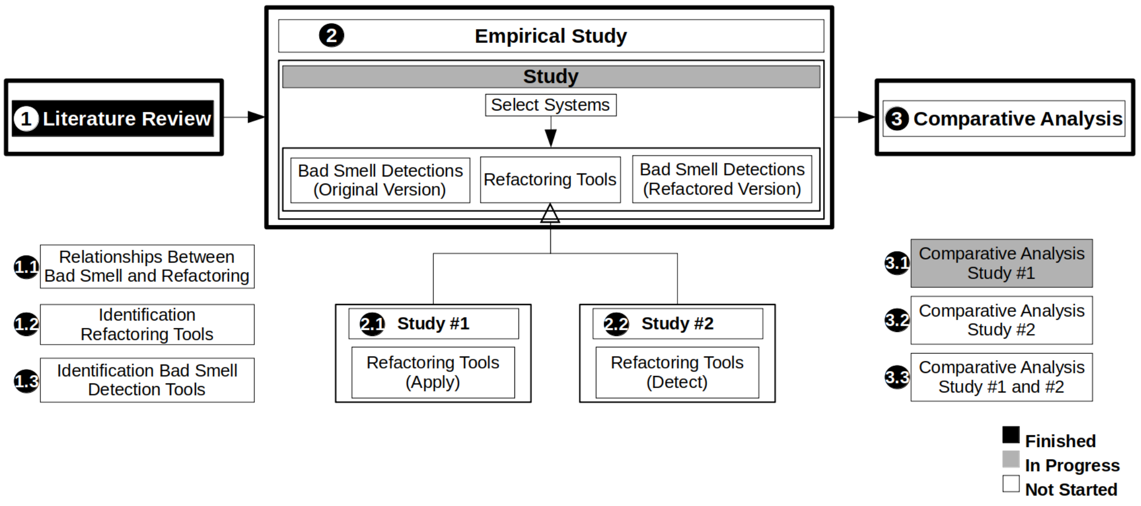
The literature presents several works discussing different approaches available to the refactoring operation [Cedrim et al. 2017, Bavota et al. 2015, Du Bois and Mens 2003, Fontana and Spinelli 2011]. For example, Bavota et al. (2015) mined the evolution history of three Java open-source projects to investigate whether refactoring activities occur on quality metrics or in bad smells, suggesting a need for refactoring operations. According to their results, quality metrics usually do not show a clear relationship with refactoring; 42% of refactoring operations are performed on code entities affected by code smells, and only 7% of the performed operations remove the code smells. Different from this work, we intend to use a larger number of Java systems. Furthermore, we aim not only to identify the code entities affected by refactoring but also to present which bad smells have been removed and introduced by refactoring.

Cedrim et al. (2017) analyze how 16,566 refactorings distributed in ten different types affect the density of 13 types of code smells in the version histories of 23 projects. Their results reveal that 79.4% of the refactorings touched smelly elements, 57% did not reduce their occurrences, 9.7% of refactorings removed smells, and 33.3% induced new ones. They also characterized and quantified that 30% of the Move Method and Pull Up Method induced God Class's emergence, and the Extract Superclass in 68% of the cases create Speculative Generality. Unlike this study, we focused only on the investigation of the effects of refactorings on Fowler's bad smells. Besides detecting refactorings that have been carried out in the source code, we intend to apply refactorings using tools to incorporate different manual or automated refactoring strategies. Finally, as one of the results, we will provide a catalog showing which bad smells we may remove and which ones, if they exist, we may introduce by refactoring.

3. Research Agenda

The goal of this project is to identify the effects of the refactoring operation on bad smells. These effects will be measured by detecting bad smells in an original version of the system and after applying a specific refactoring operation. After performing the detections of bad smells in both versions, original and refactored, we will perform the comparative analysis results and find the refactoring's effects. Figure 1 exhibits the diagram of steps planned to conduct the complete research. We divide our project into three stages: Literature Review, Empirical Study, and Comparative Analysis. Each stage may be *finished*, *in progress*, or *not started yet*. To discuss the process planned to conduct each stage and its progress in this section's remainder, we will use only the step number under parenthesis inside each stage, referring to Figure 1.

Literature Review. In this stage, we conducted a literature review to identify what the literature already discussed on three subjects: bad smell tools, refactoring tools, and the theoretical relationship between bad smells and refactoring. To find the refactoring tools presented in the literature, we conducted a Study Mapping in Step 1.2



2 Empirical Study
Study
Select Systems
Bad Smell Detections (Original Version)
Refactoring Tools
Bad Smell Detections (Refactored Version)
1 Literature Review
3 Comparative Analysis
1.1 Relationships Between Bad Smell and Refactoring
1.2 Identification Refactoring Tools
1.3 Identification Bad Smell Detection Tools
2.1 Study #1
Refactoring Tools (Apply)
2.2 Study #2
Refactoring Tools (Detect)
3.1 Comparative Analysis Study #1
3.2 Comparative Analysis Study #2
3.3 Comparative Analysis Study #1 and #2
Finished
In Progress
Not Started

Figure 1. Diagram of steps to conduct

and identified 44 refactoring tools proposed from 2013 up to 2017 [Tavares et al. 2018]. For the bad smell tools, Step 1.3, we used studies already presented in the literature [Sobrinho et al. 2018, Fernandes et al. 2016]. To identify the relationship between Bad Smells and Refactoring discussed in the literature, we conducted a Systematic Literature Review in Step 1.1 and identified 20 papers that showed the direct relationship between 31 different refactorings and 16 bad smells [Tavares et al. 2020b]. Surprisingly, we identified relationships not discussed in the Fowler's catalog. Finally, we highlighted the refactoring strategies that we may perform to eliminate bad smells. We have done all the steps, therefore, finishing this stage.

Empirical Study. In this stage, we are planning to conduct two studies, as shown in Figure 1. In the first study, Step 2.1, we performed a refactoring strategy in an automated way and evaluated their effects. Initially, we defined the systems, the bad smells, and refactoring tools we will use. After that, we selected a specific system in their original version and detected all bad smells present in the source code. After the detections on the original version, we applied the refactoring strategy. Finally, we detected the bad smells again; but now in the refactored version. Furthermore, we performed a comparative analysis with both bad smell detections and detected the automated refactoring effects. Step 2.1 is composed of eight systems, five bad smells tools, and one refactoring tool. To compose a complete database, we intend to extend Step 2.1 to more tools and systems. In the second study, Step 2.2, we plan to conduct a similar study of the first one. Unlike Step 2.1, we want to use tools to detect refactoring already applied in in this new study's source code. To conduct it, we will evaluate two versions of a system, detect the bad smells and refactorings between them. We did not start the second study yet.

Comparative Analysis. To finish the Comparative Analysis, Stage 3, we need the results produced by Stage 2 to assess the effects of refactoring operation on bad smells, detecting them before and after the applied refactoring. With these detections, we will conduct a comparative analysis and consequently detect the bad smells that tend to be solved or introduced by refactoring. In the first study of Stage 3, Step 3.1, we conducted the comparative analysis, and surprisingly, we identified that the automated refactoring operation might introduce bad smells in the source code. By extending the database of this step based on the extension of Step 2.1, we hope to build a complete base to support our findings. With both studies of Stage 2, we hope to be able to cover possible ways of applying refactoring, whether manual or automatic. The first study, Step 2.1, focuses only on refactorings applied as an automatic way, and the second one, Step 2.2, may detect refactoring performed in both manual and automatic ways. We believe that with the results provided by these studies, we will be able to generalize all types of refactorings that may be applied.

Furthermore, with our results, we intend to provide a complete catalog with all effects of the refactoring operation on bad smells found in these empirical evaluations. The comparative analysis of Step 3.1 is in progress, remaining the database extension. The second study, Step 3.2, and the merge of results provided by both studies, Step 3.3, have not started yet.

4. Preliminary Results

We have already published a paper as a mapping study to detect refactoring tools discussed in the literature [Tavares et al. 2018]. We identified 44 refactoring tools, which we characterized and summarized some of their features. We found 37 that offer support for Java language, 17 of the tools found are *plugins*, 4 are prototypes, and 21 refactoring tools already offer their operations to be performed in an automated way. Finally, we identified that the refactoring tools offer excellent support for refactoring strategies of the type *Move, Rename, Pull Up, Extract,* and *Clone*.

In 2020, we have a second paper accepted in an IberoAmerica conference. This paper presents a Systematic Literature Review about the relationship between bad smells and refactorings [Tavares et al. 2020b]. We found 20 different papers that show the direct relationship between 31 refactoring types and 16 bad smells proposed by Fowler. We also found seven tools that apply refactoring after detecting bad smells. We identified that the most discussed relationship in the literature is between Move Method and Feature Envy. It also revealed that there are different refactoring strategies than those discussed by Fowler to address bad smells. The literature addresses the most strategies defined in Fowler's book, and shows that most refactoring tools do not detect bad smells.

Finally, as our last result, we have a paper accepted, presenting our empirical study about the effects of the automated refactoring operation [Tavares et al. 2020a]. To conduct this research, we selected a sample of Qualitas Corpus systems [Tempero et al. 2010]. We applied two refactorings and measured their effect on ten different bad smells detected by five different tools. We observed that the two types of refactorings generate a decrease, increase, and neutral variations in the number of bad smells. Unlike Fowler's definition, we surprisingly found that decreasing cases was the lowest compared to the others. Replace Refactoring was the one presenting the lowest decrease case (0.75%) while Move Method showed the highest decrease (13.53%). To better help developers, we investigated which bad smells tend to be introduced and removed by refactoring. For instance, the systems analyzed by Move Method refactoring removed 85.71% Feature Envy.

5. Conclusion

With our preliminary results, we achieved different contributions. First, it shows a catalog of refactoring tools recently published in the literature [Tavares et al. 2018]. Second, identifying some tools that refactor through the identification of Fowler's bad smells, the relationship between bad smells and refactoring discussed in the literature, and a contrast between the literature and Fowler's catalog [Tavares et al. 2020b]. Moreover, in the last work submitted, our contributions were a comparative study that prioritizes detections performed by five bad smell tools, a catalog that presents which bad smells are introduced or solved by the refactoring operation, results of an evaluation exhibiting the effect of the refactoring operation on detecting bad smells.

Finally, with the completion of this master project, we expect to contribute to the software engineering community in the following ways:

- provide evidence to confirm or deny the findings discussed in the literature
- understand what the effects of refactoring operation cause on bad smells
- provide a catalog showing which bad smells tend to be introduced and removed by the refactoring strategy.

Besides that, we expect that the contributions mentioned assist developers in different ways, for instance:

- developers who want to apply refactoring strategies, automatically or manually, will be better informed of which bad smells may be introduced or removed by the refactoring operation
- assist developers in taking care and performing the most efficient and robust refactoring tools to not introduce new bad smells in the source code.

References

Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., and Palomba, F. (2015). An experimental investigation on the innate relationship between quality and refactoring. *Journal of Systems and Software*, pages 1–14.

Boussaa, M., Kessentini, W., Kessentini, M., Bechikh, S., and Chikha, S. B. (2013). Competitive coevolutionary code-smells detection. In *International Symposium on Search Based Software Engineering*, pages 50–65.

Cedrim, D., Garcia, A., Mongiovi, M., Gheyi, R., Sousa, L., de Mello, R., Fonseca, B., Ribeiro, M., and Chávez, A. (2017). Understanding the impact of refactoring on smells: A longitudinal study of 23 software projects. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*, pages 465–475.

Du Bois, B. and Mens, T. (2003). Describing the impact of refactoring on internal program quality. In *International Workshop on Evolution of Large-scale Industrial Software Applications*, pages 37–48.

Fernandes, E., Oliveira, J., Vale, G., Paiva, T., and Figueiredo, E. (2016). A review-based comparative study of bad smell detection tools. In *20th International Conference on Evaluation and Assessment in Software Engineering*, pages 1–12.

Fokaefs, M., Tsantalis, N., Stroulia, E., and Chatzigeorgiou, A. (2011). Jdeodorant: identification and application of extract class refactorings. In *33rd International Conference on Software Engineering (ICSE)*, pages 1037–1039.

Fontana, F. A. and Spinelli, S. (2011). Impact of refactoring on quality code evaluation. In *Proceedings of the 4th Workshop on Refactoring Tools*, pages 37–40.

Fowler, M., Beck, K., Brant, J., and Opdyke, W. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

Kataoka, Y., Imai, T., Andou, H., and Fukaya, T. (2002). A quantitative evaluation of maintainability enhancement by refactoring. In *International Conference on Software Maintenance*, pages 576–585.

Kessentini, M., Vaucher, S., and Sahraoui, H. (2010). Deviance from perfection is a better criterion than closeness to evil when identifying risky code. In *Proceedings of the IEEE/ACM Int. Conference on Automated Software Engineering*, pages 113–122.

Mkaouer, M. W., Kessentini, M., Bechikh, S., Cinnéide, M. Ó., and Deb, K. (2016). On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach. *Empirical Soft. Eng.*, pages 2503–2545.

Moha, N., Gueheneuc, Y.-G., Duchien, L., and Le Meur, A.-F. (2009). Decor: A method for the specification and detection of code and design smells. *IEEE Transactions on Software Engineering*, pages 20–36.

Moser, R., Sillitti, A., Abrahamsson, P., and Succi, G. (2006). Does refactoring improve reusability? In *International Conference on Software Reuse*, pages 287–297.

Pietrzak, B. and Walter, B. (2006). Leveraging code smell detection with inter-smell relations. In *International Conference on Extreme Programming and Agile Processes in Software Engineering*, pages 75–84.

Sobrinho, E. V., De Lucia, A., and Maia, M. (2018). A systematic literature review on bad smells—5 w's: which, when, what, who, where. *IEEE Transactions on Software Engineering*.

Tavares, C., Bigonha, M., and Figueiredo, E. (Accepted in 2020a). Analyzing the impact of refactoring on bad smells. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering (SBES)*.

Tavares, C. S., Ferreira, F., and Figueiredo, E. (2018). A systematic mapping of literature on software refactoring tools. In *Proceedings of the XIV Brazilian Symposium on Information Systems*, page 11.

Tavares, C. S., Santana, A., Figueiredo, E., and Bigonha, M. A. S. (Accepted in 2020b). Revisiting the bad smell and refactoring relationship: A systematic literature review. In *Experimental Software Engineering (ESELAW)*.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., and Noble, J. (2010). Qualitas corpus: A curated collection of java code for empirical studies. In *Asia Pacific Software Engineering Conference (APSEC)*, pages 336–345.

Tsantalis, N. and Chatzigeorgiou, A. (2009). Identification of move method refactoring opportunities. *IEEE Transactions on Software Engineering*, pages 347–367.