On the Impact of Bad Smell Agglomerations on Software Quality

Amanda Damasceno Santana[1], Eduardo Figueiredo[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais – Belo Horizonte, MG - Brazil
{amandads,figueiredo}@dcc.ufmg.br

Abstract. *When a system evolution is not planned, developers can take decisions that degrade the system quality. To cope with this problem, refactoring can be applied to the source code aiming to increase code quality without modifying the software external behavior. To know when to refactor, the concept of bad smells can be used. Bad smells are snippets of source code that suggest the need of refactoring. However, bad smells does not always appear isolated. The aim of this study is to understand the impact of bad smell agglomerations on the software quality by evaluating a large dataset of open source systems. To achieve our goal, we plan to use data mining techniques complemented with correlation analysis of the dataset.*

Keywords: Bad Smell, Refactoring, Agglomeration, Software Quality.

Level: Master Degree
Pos Graduation on Computer Science at Federal University of Minas Gerais
Entry year: second semester of 2018
Expected day of defense: first semester of 2020
Approval of project: submitted in May 2019
WTDSoft

1. Introduction

Software systems must evolve to cope with new requirements of the customers and changes in the environment where they have been deployed. This evolution increases the complexity of the source code, leading developers to make sub-optimal design decisions [van Gurp and Bosch 2002]. Design problems, such as bad use of design patterns or simply a long chain of method calls, can be reduced using the process of refactoring [Kerievsky 2005]. According to Fowler (1999), *refactoring* is the process of modifying the source code in such a way that preserves its behavior, but increases its quality. These modifications can involve, for example, removal of code duplication or simplifying the complexity of a code snippet [Kerievsky 2005].

However, know when to refactor is a challenging task for many developers. To help developers in this task, Fowler (1999) provided in their book a list of 22 *bad smells*. They are structures in the source code that needs to be refactored, indicating a design problem. Many authors tried to understand the impact of bad smells on the software quality, in terms of change proneness and fault proneness [Palomba et al. 2018] [Khom et al. 2012], maintenance effort [Sjøberg et al. 2012] [Yamashita 2014] and perception of the developers and students about the harm that the bad smells can present [Palomba et al. 2014] [Yamashita and Moonen 2013] [Abbes et al. 2011].

However, a single snippet of code (for example a class or method) can have more than one bad smell, making the process of refactoring even harder. When a piece of code presents more than one bad smell, called *bad smell agglomeration*, a group of bad smells are said to be *inter related* [Oizumi et al. 2016]. If a single bad smell requires different refactoring operations to be removed, an agglomeration can be a much harder challenge, because each bad smell will have to be refactored (one at a time) using the operation that removes it, being necessary to check if a bad smell removal impacts directly on other bad smells that belong to the agglomeration.

The focus of this Master's project is to give an insight on the impact of these bad smell agglomerations, giving initial evidences of which agglomerations needs to be prioritized on the refactoring process, i.e. classes that are more prone to be changed in future versions of the system. For this purpose, we plan to analyze bad smell agglomerations in a large dataset composed of open source systems of different sizes and domains, considering them at the level of a class. For example, if a class **C** presents bad smells of type **a**, **b**, and **c**, it is an indicative of the presence of a bad smells agglomeration in class **C**. The fact that certain types of bad smells appears at different levels of granularity, such as method and attributes, will be considered as a mean of understanding if the agglomeration of bad smells impacts directly on the existence of bad smell at the class level.

As results, we expect to find the following novelties: tuples of bad smells that appear together in the source code, classifying them to understand the nature of the relationship, for example, if bad smell **a** makes impossible the existence of smell **b** in the same class; finally, through refactoring mining and metrics calculation, the impact on change-proneness, on complexity, cohesion, and coupling of such bad smells agglomerations will be studied. To complement the analysis of change-proneness it will be verified if such agglomerations were refactored or not, giving an initial idea of which agglomerations needs to be prioritized in the process of refactoring.

2. Theoretical Background

According to a recent systematic literature review on bad smells [Sobrinho et al. 2018], most studies in the white literature do not establish a relationship between bad smells. Another finding is that there is a focus on only five of the 22 bad smells defined by Fowler, indicating the need of more research to both deep and expands the current knowledge on bad smell agglomerations. The bad smells are: Duplicated Code, Large Class, Feature Envy, Long Method and Data Class. Also, the authors found that for each of these five bad smells, they are mostly co-studied among them, with the predominance of co-studies with the Large Class smell. So the focus of this project is to evaluate bad smells that were not studied deeply, such as Divergent Change, Speculative Generality and Refused Bequest, trying to identify their impact and relationship with bad smells that were already studied.

As the literature suggests, the presence of bad smell agglomerations can be harmful to the system quality. For instance, Palomba et al. (2018) found that the presence of more than one bad smell in code increases the change and fault proneness. Yamashita and Moonen (2013) found evidences that some (not all) agglomerations can difficult the maintenance process. Lozano et al. (2015) focused on raising evidences that the co-existence of bad smells degrade the system quality. However, most of these studies focus on only discovering and classifying the relationships in small dataset, evidencing the need of research that evaluates the impact of the agglomerations on the source code on a larger dataset, exploring the consequences of the refactoring process.

This project also relies heavily on the classification of bad smell relations proposed by Pietzrak and Walter (2006). We believe this classification allows a deep understanding of bad smell agglomerations and their impact on the software quality, because the classification allows identifying the type of relationship that the bad smells in the agglomeration have. For example, in their work, Pietzrak and Walter (2006) found that 90% of the Large Class analyzed presented at least one Feature Envy, suggesting the relationship of plain support.

Based on this classification, we consider the following categories of relationships between bad smells: Plain Support is when a bad smell **a** implies the existence of a bad smell **b**; Mutual Support is when both smells **a** and **b** support each other, implying that they originate from the same design flaw; Rejection means that the presence of smell **a** implies that smell **b** does not exist in the same entity; Aggregate Support is when the presence of a set of bad smells **C**, in which $C = \{ c_1, c_2, \ldots, c_n\}$ and $\mathbf{c_i}$ is a smell belonging to C, C implies the existence of a smell **d**; Transitive Support means if a smell **a** supports smell **b**, and smell **b** supports smell **c**, then **a** supports **c**; Inclusion: smell **a** is a particular case of smell **b**.

3. Expected Contributions

This project aims at contributing to the software engineering community in the following ways:

- Understand the relationship between bad smells that were not yet co-studied in the literature;

- Provide evidences to confirm or deny the findings of other authors (discussed in the Related Work section) about the relationship between bad smells;
- Understand the impact of bad smells agglomerations on change-proneness and if these agglomerations are in fact removed when the process of refactoring occurs;
- Understand the impact of these agglomerations on cohesion, complexity, and coupling;
- Provide a public dataset of bad smells agglomerations in open source systems, making it available in the GitHub plataform.

4. Methodology

This project aims at deep understanding the relationship between bad smells at the level of class in a large dataset. These relationships will be classified according to their type based on the classification presented in Section 2. This analysis aims to identify types of relationships and agglomerations that mostly impact on the quality of the source code. Figure 1 presents a process diagram with the steps planned for my master research and the color of the circles shows if the step was concluded or is a work in progress.

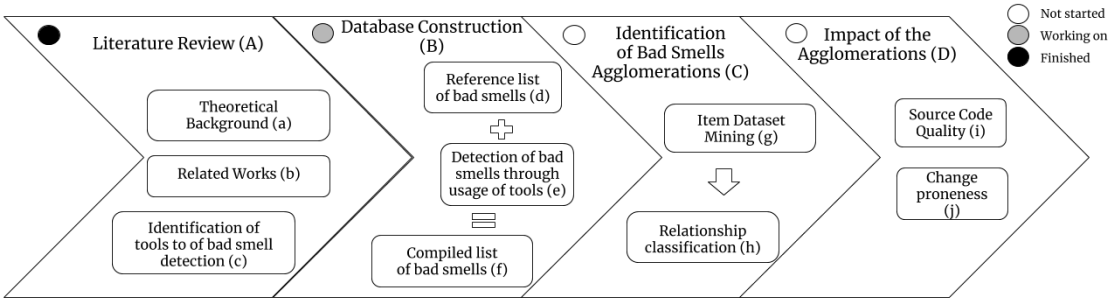

Figure 1. Diagram that summarize the steps taken in this project

The first step (A) aims to identify the gaps in the literature, in an nonsystematic fashion, the bad smell agglomerations that were already studied and had its impact evaluated, tools that can be used to create the oracle of bad smells, and the theoretical background to support this project. First (a), it was searched for studies on a relevant database that: tries to conceptualize, or categorize bad smells; and analyzes their impact on the source code such as change proneness and quality. The findings in this phase allowed constructing a robust motivation for this project.

The second phase (b) was to search for studies about agglomerations, inter-smell relations, co-study, and co-occurrence of bad smells. This phase allowed the identification of: what bad smells were studied; systems that were used; methodology that were applied to identify such relations; and how the oracles of bad smell instances were created. This phase allowed the definition of which methodology to apply to this project and what bad smells to be evaluated. Finally, the last phase (c) in this step (A) was to identify tools that detect bad smells. Previous literature reviews [Fernandes et al. 2016] [Sobrinho et al. 2018] identify the most frequent bad smell detection tools used in the literature, allowing the identification what tools to be used to detect the selected bad smells.

With the gaps identified and all background needed, we define the problem scope. The first problem in the identification of agglomerations is to create a dataset that is large enough to present meaningful relationships (B). We are going to be using the systems present at Landfill (**d**), a Web-Based dataset of bad smells [Palomba et al. 2015], and enriching it with the assistance of detection tools (phase **f**). We opt for Landfill because it is accepted by the community and most systems in this dataset are among the most used in the bad smell area [Sobrinho et al. 2018].

Although Landfill has identified 243 bad smell instances, they are only in five types; three of them already highly studied in the literature [Walter et al. 2018] [Palomba et al. 2018] [Yamashita et al. 2015]. In addition, most of the studies in the literature focus on a small number of systems, and use only one tool to detect the bad smell. Hence, there is a need to create a larger dataset that contains more instances of different kinds of smells in systems of different domains and sizes.

To create this dataset, for each bad smell under study, illustrated in the second step (e) in Figure 1, we will run two or three tools on the dataset of systems used by the bad smell community. The tools to be used are the most well-known and reliable ones [Sobrinho et al. 2018]. Each tool will give a vote to the presence of the bad smell detected. If two of the tools agree that a class has a bad smell, the class and the bad smell detected is added to the dataset. Further, to give robustness to our dataset, for each bad smell under study, it will be calculated the agreements between tools on the true positives instances.

The next step is to obtain the agglomerations of bad smells that are more meaningful in our constructed dataset (G). For this purpose, it might apply the Item Dataset Mining technique (g). This technique was proposed by Agrawal et al. (1993) as a simple way to extract meaningful relationships from a large database through the identification of the antecedents and consequents. We can use this technique to identify pattern of items that appears together in the transactions, in our case the bad smells, these being represented as association rules. To support this step we will use as a guideline the work of Halkidi et al. (2011) that presents an overview of data mining techniques applied to software engineering.

However, not all relationships are interesting. To determine what is meaningful, it will be used the measurements of Support, Confidence, Correlation [Han et al. 2011], Lift, Leverage, Item Set-Lift and Item Set-Leverage. These measurements allow determining the rule's strength. With the meaningful associations rules detected, we can classify them using the classification described in Section 2, providing more information on the types of rules that are more frequent in the source code (phase h) and identify which rules will have their impact analyzed.

Finally, in the last step (D), we can analyze the impact of these relationships on the quality of the source code. AlOmar et al. (2019) studied the impact on metrics of source code when a refactor process was done in order to improve the quality of the source code. They found with statistical significance metrics that have their value modified when the developer tries to improve the cohesion, complexity and coupling of the code. These metrics provide us with a quantitative visualization of factors that affects the quality of the source code of a system (i).

To calculate the change-proneness (j) of a class affected by bad smell agglomerations, we aim to rely on two versions of the same system to verify if the class with the agglomerations was modified. For this purpose, it will be calculated the number of changes that was made to the class containing the agglomeration in the two versions of the system, allowing to verify which types of agglomerations changes with higher frequency and with statistical significance. To deep the understanding of this changing, it will be used the RefDiff tool [Silva 2017] to mine the refactoring operations that were made by developers in the two versions of the systems. This allows verifying if the classes that have agglomerations are being totally, partially or not being refactored.

5. Related Work

This section aims at describing and comparing the related work used to define the scope of this project. Pietrzak and Walter (2005) identified and classified the agglomerations of bad smells through the use of data mining techniques, which was adopted by the authors. Later (2006), the authors extended their classification. Different from the authors, our focus is on the discovery of frequent tuples/classifications in a large number of systems; i.e., not proposing new classifications. More recently, Walter et al. (2018) evaluated collocated smells in the Qualita Corpus [Tempero et al. 2010]. This work is similar to what we plan; however, ours includes an evaluation of the impact of the found relationships on the quality of the source code.

Yamashita et al. (2015) evaluated the presence of bad smells in open source and industrial system. The authors discovered the existence of redundant components, in which a certain bad smell is dependent of the same type of bad smells. Different of this study, we focus on open source systems and the use of data mining techniques to identify and classify the found relationships. Oizumi et al. (2018) proposed and evaluated Organic, a tool for the identification of bad smell agglomerations. Here, we are trying to give evidence on which agglomerations mostly impact on quality.

References

Gurp, J. V. and Bosch, J. (2002) “Design erosion: problems and causes,” Journal of Systems and Software, vol. 61, no. 2, pp. 105 – 119.

Fowler, M. and Beck, K. (1999) "Refactoring: Improving the Design of Existing Code," Object Technology Series. Addison-Wesley.

Kerievsky, J. (2005) "Refactoring to Patterns," The Addison-Wesley Signature Series. Addison-Wesley.

F. Palomba, et al. (2018) "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," Empirical Software Engineering 23: 1188.

Khomh, F. et al. (2012) “An Exploratory Study of the Impact of Antipatterns on Class Change- and Fault-proneness,” Empirical Software Engineering, vol. 17, no. 3, pp. 243–275.

Sjøberg, D. I. K. et al. (2013) "Quantifying the Effect of Code Smells on Maintenance Effort," in IEEE Transactions on Software Engineering, vol. 39, no. 8, pp. 1144-1156.

Yamashita, A. (2014) "Assessing the capability of code smells to explain maintenance problems: An empirical study combining quantitative and qualitative data", in Empirical Software Engineering, 19.

Palomba, F. et al. (2014) “Do They Really Smell Bad? A Study on Developers’ Perception of Bad Code Smells,” in Int. Conf. on Software Maintenance and Evolution, pp. 101–110.

Yamashita, A. and Moonen, L. (2013) “Do developers care about code smells? An exploratory survey,” in Working Conf. on Reverse Engineering. IEEE, pp. 242–251.

Sobrinho, V. de P. et al. (2018) "A systematic literature review on bad smells — 5 W's: which, when, what, who, where". IEEE Transactions on Software Engineering.

Pietrzak, B. and Walter, B. (2006) “Leveraging Code Smell Detection with Inter-smell Relations,” in Proc. of the 7th Int. Conf. on Extreme Programming and Agile Processes in Software Engineering. Springer-Verlag, pp. 75–84.

Palomba, F. et al. (2015) “Landfill: An open dataset of code smells with public evaluation,” in IEEE/ACM 12th Working Conf. on Mining Software Repositories, pp. 482–485.

Fernandes, E. et al. (2016) “A review-based comparative study of bad smell detection tools,” in Proc. of the Int. Conf. on Evaluation and Assessment in Software Engineering, pp. 1–12.

Walter, B. et al. (2018) "Code smells and their collocations: A large-scale experiment on open-source systems," Journal of Systems and Software, vol. 144, pp. 1-21.

Yamashita, A. et al. (2015) “Inter-smell relations in industrial and open source systems: A replication and comparative analysis,” in IEEE Int. Conf. on Software Maintenance and Evolution pp. 121–130.

Agrawal, R. et al. (1993) “Mining Association Rules Between Sets of Items in Large Databases,” in SIGMOD Conference.

Han, J. et al. (2011) “Data Mining: Concepts and Techniques,” Morgan Kaufmann Publishers Inc., 3rd edition.

Pietrzak, B. and Walter, B. (2005) “Exploring Bad Code Smells Dependencies,” in: Proc. of the Conf. on Software Engineering: Evolution and Emerging Technologies, pp. 353-364.

Walter, B. et al (2018) "Code smells and their collocations: A large-scale experiment on open-source systems," Journal of Systems and Software, Volume 144, Pages 1-21.

Tempero, E. et al. (2010) "Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies", Asia Pacific Software Engineering Conference, pp. 336–345.

Abbes, M. et al. (2011) “An Empirical Study of the Impact of Two Antipatterns, Blob and Spaghetti Code, on Program Comprehension,” in 15th European Conference on Software Maintenance and Reengineering. IEEE, pp. 181–190.

Oizumi, W. et al. (2018) "On the identification of design problems in stinky code: experiences and tool support," in J Brazilian Computer Society 24: 13.

Oizumi, W. et al. (2016) "Code anomalies flock together: exploring code anomaly agglomerations for locating design problems," in Proc. of the 38th Int. Conf. on Software Engineering, pp. 440-451.

Halkidi, M. et al. (2011) “Data mining in software engineering,” in Intelligent Data Analysis, pp. 413-441.

AlOmar, E. A. et al. (2019) “Do Design Metrics Capture Developers Perception of Quality? An Empirical Study on Self-Affirmed Refactoring Activities,” in Empirical Software Engineering.

Silva, D. and Valente, M. T. (2017) “RefDiff: Detecting Refactorings in Version Histories,” in Int. Conf. on Mining Software Repositories.