Revisiting the Bad Smell and Refactoring Relationship: A Systematic Literature Review

Cleiton Silva[1], Amanda Santana[1], Eduardo Figueiredo[1], Mariza A. S. Bigonha[1]

Department of Computer Science, Federal University of Minas Gerais,
Belo Horizonte, Brazil
{cleiton.silva, amandads, figueiredo, mariza}@dcc.ufmg.br

Abstract. Bad smells are indicators of code structure problems that may be solved via refactoring. Refactoring is a process of improving the internal structure of the source code without changing the external behavior of the software system. However, the refactoring process takes effort and, so, it should be done in a disciplined manner in order to minimize the chances of introducing code errors. The goal of this paper is to present a systematic literature review with the purpose of identifying the explicit relationship between refactorings and the bad smells proposed by Fowler. This relationship is defined as the environment in which a bad smell is mentioned and a possible refactoring strategy that may be used to solve it. As a result, we have identified 20 papers that show the direct relationship between 31 different refactorings and 16 bad smells. Among the contributions of our SLR, we highlight the refactoring strategies that may be performed to eliminate bad smells; the relationship between bad smells and refactoring discussed in the literature up to now; and a contrast found between the literature and Fowler's catalog.

Keywords: Bad Smell · Code Smell · Refactoring · Literature Review

1 Introduction

Software systems must evolve to cope with new requirements from stakeholders. Hence, it requires a great effort for developers to understand and make the necessary modifications in the source code. This effort is greatly affected by aspects of the quality of the source code, such as comprehensibility, complexity, and maintainability. *Bad Smells* are indicators that there are code structure problems, which may be solved via refactoring [19]. *Refactoring* is a process of enhancing the internal structure of the source code without changing the external behavior of the software system [18].

Fowler [18] presents a catalog of bad smells and refactoring consisting of 22 bad smells and 72 refactorings, which is widely discussed in the literature [29, 33, 44]. However, in many cases, the descriptions of bad smells and their relationships with refactorings are not precise and lack a consistent level of details [4]. So, we need more studies in the area, not only in context-based evaluations of bad smells and on how to eliminate them [45], but also in research exploring the relationship between refactoring and bad smell [9].

Studies that perform analysis based on the refactoring process argue that refactoring is effective in removing bad smells in less than 10% of the time [5, 10, 11]. So, refactoring should be done in a disciplined manner to minimize the chances of introducing (i) another bad smell [10, 11, 30, 48]; (ii) other defects [6, 32, 40]; and (iii) ensure that the quality was enhanced [5, 18]. Xia et al. [53] have conducted an empirical study by surveying with developers, and they have identified the widely use of refactoring. However, participants reported that this activity is often neglected in academia, which focuses on the ability to write code from scratch. Therefore, they suggest that educators should emphasize refactoring in classes in order to identify and eliminating bad smells.

Refactor is a tricky activity since developers have to analyze the source code to (i) identify what will be refactored, (ii) the operation that will solve it, and (iii) where the refactored code will be allocated. These steps are difficult to be taught since they depend on the system context, although they may be addressed using automatic refactoring tools. Some developers do not use these tools because they do not know when and how to refactor [26]. In fact, some studies reveal that many developers prefer to refactor code manually, because they do not fully trust the tool behavior [23]. They also perform refactoring manually because existing refactoring tools may introduce new bad smells after the refactoring process [50].

To address the problems mentioned, this paper presents a systematic literature review (SLR) whose objective aims at finding direct relationship between the bad smell and the refactoring proposed in the Fowler's catalog [18]. We have identified 20 papers that show the explicit relationship between 31 different refactoring and 16 bad smells. Analyzing these papers, we have found that (1) the relationship between Move Method applied to Feature Envy appears as the most discussed one in these studies; (2) there are different refactoring strategies than those proposed by Fowler to address some bad smells. We also have found seven tools that refactor through the identification of the bad smells of Fowler. These results provide insights to researchers and developers of the existing relationships that may be used to refactor code. We also identify relationships that should be more investigated [52].

The main contributions of this SLR are four-fold. First, it shows what refactoring strategies may be performed to eliminate bad smells. Second, it reports the relationship between bad smells and refactoring discussed in the literature up to now. Third, it provides a contrast between the literature and Fowler's catalog. Finally, it help us to identify tools that refactor through the identification of Fowler's bad smells.

The remainder of this paper is organized as follows. Section 2 describes the SLR protocol we have followed. Section 3 presents the results of the conducted SLR. Section 4 discusses threats to validity. Section 5 discusses related work. Section 6 concludes this study and points directions for future work.

2 Systematic Literature Review

A Systematic Literature Review is a study that provides identification, analysis, and interpretation of evidence related to a particular research topic [52]. This work is conducted using the Kitchenham guidelines [24]. It has three stages: (i) planning, (ii) execution, and (iii) analysis [24]. The focus of this SLR is on the explicit relationship between refactoring and bad smells. Section 2.1 presents the planning phase. Section 2.2 describes the execution and the steps for the selection of primary studies. The analysis stage is presented in Section 3.

2.1 Planning

In this phase, we define: (i) the topic to be investigated; (ii) the electronic databases used to search for papers; (iii) the search string to identify relevant studies; (iv) the inclusion and exclusion criteria to obtain primary studies; and (v) timestamp in which this work has been conducted.

Research questions. To identify the relationships between refactoring and the bad smells of Fowler [18], we have defined two general Research Questions **RQ1** and **RQ2**, and two specific ones **RQ1.1** and **RQ1.2**. These RQs allow us reaching our goals, and drive all activities we have conducted within this SLR.

RQ1 Which relationships between refactoring and bad smells are explicitly discussed in the literature?

RQ1.1 Which are the most mentioned relationships between bad smells and refactoring found in the literature?

RQ1.2 Do relationships we found differ from those that Fowler presents?

RQ2 Which tools perform refactoring from bad smell detection?

Electronic databases. There are different eletronic databases to be used in literature reviews to search for primary studies. Some studies use one database [54, 55]; others use three databases [21], six databases [3, 15, 33], seven databases [1], or even eight databases [8, 29, 49]. We use seven of the most used ones [33], as exhibits Table 1. The first column in Table 1 shows the database name, while the second one presents their respective websites.

Search string. It identifies the relevant studies in the selected databases, allowing us to answer the proposed research questions. We conducted a pilot study with search strings composed of multiple different terms and applied them in each database. We evaluated their results in each database with the purpose to identify which of the search string have reached as many studies as possible in the literature. At the end, the search string was consolidated with three terms as follows.

```
(refactor OR refactoring) AND (relationship OR correlation OR
associate) AND ("code smell" OR "bad smell" OR bug OR "anti pattern")
```

Table 1. Eletronic Databases

Database	Address	Papers Returned
ACM Digital Library	http://dl.acm.org/	113
Engineering Village	https://www.engineeringvillage.com	83
IEEE Xplore	http://ieeexplore.ieee.org/	47
Science Direct	http://www.sciencedirect.com/	07
Scopus	http://scopus.com/	70
Springer	http://link.springer.com/	2,025
Web of Science	http://apps.webofknowledge.com/	44
Total		2,389

Inclusion and exclusion criteria. We have four inclusion and two exclusion criteria to select the primary studies (see Table 2). These criteria allow classifying each study under review as a candidate to be included or excluded from the SLR.

Table 2. Inclusion and Exclusion Criteria

Inclusion Criteria	Exclusion Criteria
Written in English	< 5 pages
Published in conferences, journals, workshops and book chapters*	Thesis, dissertations, tutorials, courses and magazines issues
Available in electronic format	
Present Fowler's refactoring and bad smells	

*Papers published at conferences that appear as book chapters in digital libraries

Search source. We searched all studies published up to 2018. The search process was carried out from February 6 to 9, 2019.

2.2 Execution

This phase consists of: (i) applying the search string in the selected databases, identifying primary studies, and (ii) selecting the relevant ones found through the exclusion/inclusion criteria. Figure 1 presents the resulting number of primary studies after each step, serving as input to the next step. The check symbols indicate the studies that remained in the step, while numbers near cross symbols indicate the studies excluded in the step. At the end of the process, we identified 20 papers that fit the scope of this work. These studies were analyzed and summarized in order to collect information to answer our RQs.

Search process. Table 1 presents the primary studies returned by the search in each database, with the total of 2,389 studies, including duplicates.

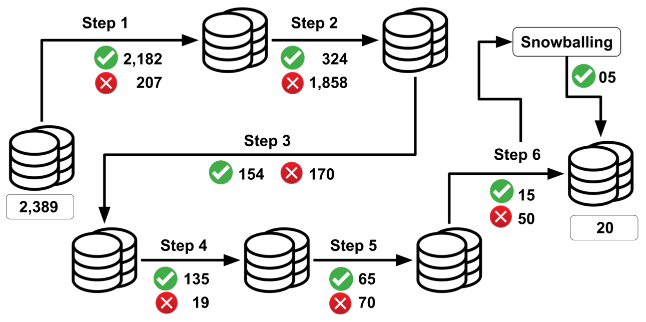

Fig. 1. Number of papers filtered in each step

Study selection process. Figure 1 presents the six steps focusing on selecting relevant papers according to its content. The result regarding each individual database is available on our website [41]. The six steps are discussed as follows.

Step 1- *Remove Duplicates.* We merge the papers returned in all databases, which cause the removal of duplicated papers. These papers were randomly eliminated, without favoring any database.

Step 2- *Reading Title.* Only those papers with excerpts of the search string used or those that could be relevant for the study are retained. In case of doubts, the papers were kept for the next filtering step.

Step 3- *Reading Abstract.* Here, we select those papers that show some evidence of being linked to the SLR context.

Step 4- *Inclusion and Exclusion Criteria.* After the application of the inclusion and exclusion criteria defined in Section 2.1, if the paper fits in at least one of the exclusion criteria, it is removed from our SLR.

Step 5- *Reading Introduction and Conclusion.* We select those papers that present evidence that they help to answer our RQs, and those that we deem relevant to the study.

Step 6- *Complete Reading of Paper.* Only those papers that are directly related to our RQs are included.

Snowballing- In our SLR, we execute the backward snowballing [51]. The process was performed with the 15 papers identified at the end of Step 6. These six steps were applied again, in order to find new candidates to be included. The snowballing has resulted in the inclusion of 5 new papers.

Data extraction. Research questions are the main drivers of what information needs to be extracted. So, the main topic of each RQ was identified and summarized in a table. For each paper found, it was documented: (i) each bad smell mentioned in the study and the refactoring operations that solve it; and (ii) each tool mentioned in the paper and their characteristics. These information allow us not only to identify the relationship that is most studied, but also to compare the Fowler suggestions against the suggestions proposed by the literature.

It also provides insights of which tools are used to study both bad smells and refactoring.

The quality assessment is an integral part of an SLR to assess the strength of the evidence, and to take it into account when synthesizing the results [2]. However, we did not perform it in this SLR. The reason is because we consider that the Study Selection Process and Data Extraction are sufficient in reaching most relevant papers to our study.

Results summarization. We found 20 studies explicitly addressing the relationship between refactoring and the bad smells proposed by Fowler [18]. It is worth noting that all selected papers were published between 2004 and 2018. This shows that the topic is still being discussed in the literature.

3 Results

This section aims to answer RQ1 and RQ2 presented in Section 2.1. Section 3.1 describes the characterization of the papers found. Sections 3.2 and 3.3 presents the analysis of the collected data to answer RQ1 and RQ2, respectively.

3.1 Overview of Primary Studies

Figure 2 shows the number of studies found per year. Observe that 2017 shows the largest number of relevant studies. The years from 2013 to 2015 show three papers each. No study discussing the explicit relationship between refactoring and bad smells was found in 2005, 2007, 2008, and 2010 to 2012. From the figure, we may argue that the topic has been recently researched.

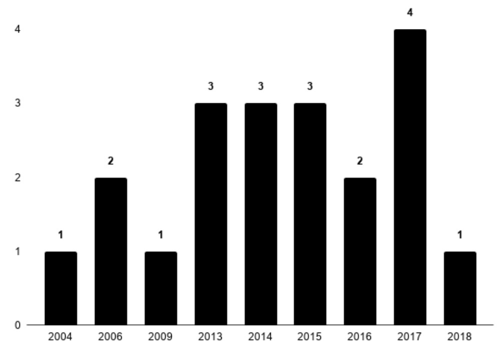

Fig. 2. Publication Year

Selected studies were mainly published in journals and conferences, with a total of 9 and 8 studies, respectively. They were published in 17 different events, most of them in the IEEE Transactions on Software Engineering and in the

International Conference on Agile Software Development (XP), with 3 and 2 papers, respectively. The remaining events presented one study each and may be found in our website [41].

3.2 Relationships between Bad Smells and Refactoring

RQ1 Which relationships between refactoring and bad smells are explicitly discussed in the literature?

Table 3 presents the relationships between refactoring and bad smells according to the analysis of the 20 papers found. Each line presents a different relationship between a bad smell and refactoring. The first column presents the bad smell, the second column shows which refactoring solves it. Columns 3 and 4 check whether the relationship is discussed by Fowler or by the literature, respectively. A bullet indicates that refactoring was discussed. It is worth noticing that a relationship may be discussed by Fowler and by the literature. The last column presents reference of the studies that discuss these relationships.

We focused on Fowler's bad smells and refactoring. But, for specific situations, we have merged different terminologies into a single bad smell, since their meaning and characteristics are similar, only changing their names. For instance, we classified Clone [27] or Code Clone [22, 28] as Duplicated Code. Parallel Inheritance [36, 40] is classified as Parallel Inheritance Hierarchies. There is a case where the authors explicitly indicated State Checking as the Switch Statements smell [12]. For the refactoring classification, we have the case where Polymorphism [12] is named for Replace Conditional with Polymorphism.

Every paper we have identified discuss more than one relationship between bad smells and refactoring. So, each relationship found is individually documented. Observe that there are cases where different refactoring may be applied to address the same bad smell. Therefore, to answer RQ1, Table 3 presents the 22 bad smells proposed by Fowler, where 16 of them (73%) are studied in the literature. This indicates concern on the part of researchers to address, as much as possible, most bad smells present in the catalog. However, the situation for refactoring is different. From a total of 72 refactoring in the catalog, only 31 (43%) of them are cited in the literature. This indicates the need to evaluate other refactoring strategies. Still, this may be an indicative that not all refactoring operations mentioned by Fowler are used in practice to remove bad smells.

RQ1.1 Which are the most mentioned relationships between bad smells and refactoring found in the literature?

Table 3 shows all relationships between refactoring and bad smells found in this SLR. To answer this RQ, we consider the relationship being discussed by the largest number of different studies. We identified that the highest number of citations is related to Move Method (refactoring) and Feature Envy (bad smell), totaling 14 different papers targeting this relationship.

According to the data presented in Table 3, the refactoring operation that solves the largest amount of bad smells is Move Method, being used to solve 11

Table 3. Relationships Between Bad Smell and Refactoring

Bad Smell	Refactoring	Fowler	Literature	References
Alternative Classes with Different Interfaces	Extract Superclass	●	●	[13]
	Move Method	●	●	[13]
	Rename Method	●	●	[13]
Comments	Extract Method	●		
	Introduce Assertion	●		
	Rename Method	●		
Data Class	Encapsulate Collection	●	●	[40, 42]
	Encapsulate Field	●	●	[13, 40, 42]
	Extract Method	●	●	[40, 42]
	Hide Method	●	●	[13, 40, 42]
	Move Method	●	●	[13, 40, 42]
	Remove Setting Method	●	●	[40]
Data Clumps	Extract Class	●		
	Introduce Parameter Object	●		
	Preserve Whole Object	●		
Divergent Change	Extract Class	●	●	[11, 36]
	Extract Method		●	[10]
	Extract Superclass		●	[11]
	Move Field		●	[11]
	Move Method		●	[11]
	Pull Up Method		●	[11]
Duplicated Code	Extract Method	●	●	[16, 22, 27, 28]
	Form Template Method		●	[16]
	Pull Up Method	●	●	[13, 16, 22]
	Replace Method with Method Object		●	[16]
	Substitute Algorithm	●		
Feature Envy	Consolidate Duplicate Conditional Frag.		●	[5]
	Extract Method	●	●	[5, 10, 40, 42, 47]
	Move Field		●	[12, 42, 47]
	Move Method	●	●	[5–7, 12, 13, 25, 28, 36–40, 42, 47]
	Pull Up Method		●	[42]
Inappropriate Intimacy	Change Bidirectional Assoc. to Unidir.	●		
	Extract Class	●		
	Hide Method	●		
	Move Field	●	●	[13]
	Move Method	●	●	[13]
Incomplete Library Class	Introduce Foreign Method	●		
	Introduce Local Extension	●		
	Move Method	●	●	[13]
Large Class	Duplicate Observed Data	●		
	Extract Class	●		
	Extract Subclass	●	●	[13]
Lazy Class	Collapse Hierarchy	●		
	Inline Class	●	●	[5]
	Move Method		●	[37]
	Push Down Method		●	[10]
Long Method	Add Parameter		●	[5]
	Consolidate Conditional Expression		●	[5, 42]
	Decompose Conditional	●	●	[42]
	Extract Method	●	●	[5, 11, 12, 20, 27, 42]
	Inline Method		●	[5]
	Introduce Explaining Variable		●	[5]
	Introduce Parameter Object	●	●	[42]
	Preserve Whole Object	●	●	[42]
	Remove Control Flag		●	[5]
	Remove Parameter		●	[5]
	Rename Method		●	[5]
	Replace Method With Method Object	●	●	[42]
	Replace Temp With Query	●	●	[42]
Long Parameter List	Replace Parameter with Explicit Methods	●		
Message Chains	Extract Method	●		
	Hide Delegate	●		
	Move Method	●	●	[13]
Middle Man	Inline Method	●		
	Remove Middle Man	●		
	Replace Delegation with Inheritance	●		
Parallel Inheritance Hierarchies	Extract Subclass		●	[36]
	Move Field	●	●	[13]
	Move Method	●	●	[13, 40]
Primitive Obsession	Extract Class	●		
	Introduce Parameter Object	●		
	Replace Data Value with Object	●		
	Replace Type Code with Class	●		
	Replace Type Code with Subclass	●		
	Replace Type Code with State/Strategy	●		
	Replace Array with Object	●		
Refused Bequest	Push Down Field	●	●	[5, 13]
	Push Down Method	●	●	[5, 10, 13]
	Replace Inheritance with Delegation	●	●	[5]
Shotgun Surgery	Move Class	●		
	Move Field		●	[11, 13, 36]
	Move Method	●	●	[11, 13, 36]
	Pull Up Method		●	[11]
Speculative Generality	Collapse Hierarchy	●	●	[5]
	Inline Class	●		
	Remove Parameter	●	●	[13]
	Rename Method		●	[13]
Switch Statement	Extract Method	●		
	Move Method	●	●	[13]
	Replace Conditional with Polymorphism		●	[12]
	Replace Type Code with State/Strategy	●		
	Replace Type Code with Subclass	●		
Temporary Field	Extract Class	●		
	Introduce Null Object	●		

types of smells. The bad smell that may be solved with the highest number of operations is the Long Method, in which 13 refactoring may solve it.

RQ1.2 Do relationships we found differ from those that Fowler presents?

To answer this question, we performed the analysis based on columns 3 and 4 of Table 3. We identified three cases: Case 1: if Column 3 is marked and Column 4 is not, this means that this relationship is discussed only by Fowler, meaning that the relationship needs to be researched, or may be not used in practice. Case 2: if the opposite situation occurs, where Column 4 is marked and Column 3 is not, it means that we found a new relationship that is not addressed in Fowler's catalog. Case 3: if columns 3 and 4 are marked, this represents that the relationship found by Fowler agrees with what the literature says.

For example, we identified for Lazy Class all refactoring operations that solve it. We found a refactoring that only Fowler presents a solution: Collapse Hierarchy (Case 1). We also found that the literature proposes two new refactoring operations to solve it: Move Method and Push Down Method (Case 2). Finally, we identify that Inline Class was found in Fowler's work and in the literature (Case 3).

From Table 3 and by the reasoning explained above, we may conclude that only Alternative Classes with Different Interfaces, Data Class, and Refused Bequest have both columns 3 and 4 being a perfect match. This indicates that there is no new refactoring strategy to solve them reported in the literature. Also, six bad smells in the catalog were not addressed by the literature. This may be due to the nature of them, since some bad smells do not impact significantly on the source quality [35]. Observe that for seven of the 22 smells, the literature has proposed more refactoring operations than Fowler. However, for these smells, Fowler has presented 18 refactorings strategies to solve them, of which only three of them were not confirmed by the literature. We identified 35 relationships in Case 1, 24 in Case 2, and 35 in Case 3.

3.3 Tools from Refactoring

RQ2 Which tools perform refactoring from bad smell detection?

It is worth noticing that not all tools that were proposed or mentioned in the literature answer the **RQ2**. The focus of this SLR is to identify studies that explicitly cite the relationship between refactoring and bad smells, and not to identify all refactoring tools. So, we documented only tools that appear in the resulting studies, which detect bad smells, and propose refactoring to solve them. Out of 20 studies, 16 have mentioned the tools used to conduct them. This list may be found on our website [41]. We observe that nine tools need more than one tool to perform the process of detecting bad smells and applying the refactoring. As the focus of our study is to tackle the refactoring and bad smell context together, we only report seven tools that target both concepts. Table 4 presents these tools and information about each tool, such as if it has an Graphical User Interface, or if it is commercial.

Table 4. Characteristics each tool supports

Tool [Ref.]	GUI	FRA	ONL	PLG	FRE	OPS	USG	SL
Extract Method Detector [27]	Yes	-	No	Yes	Yes	Yes	No	Java
JDeodorant [6, 12, 40]	Yes	-	No	Yes	Yes	Yes	Yes	Java
JMove [39]	Yes	-	-	Yes	Yes	Yes	Yes	-
Liu's Approach [25]	-	-	-	-	-	-	-	-
Methodbook [7]	-	-	-	-	-	-	-	Java
MMRUC3 [38]	-	Yes	-	-	-	-	-	Java
Tsantalis's Methodology [47]	-	-	-	-	-	-	-	Java

GUI: graphical user interface; **FRA**: framework; **ONL**: online; **PLG**: plugin; **FRE**: free for use; **OPS**: open-source; **USG**: user guide available; **SL**: supported language; **"-"**: information not available.

4 Threats to Validity

We discuss the threats to validity listed by Wohlin et al. [52]: construct, internal, conclusion, and external.

Construct Validity: the electronic database selected might not retrieve all relevant papers. To minimize this threat, we selected seven different databases that aggregate papers from many publishers. Also, the search string used may not find all papers that are relevant to this SLR. To minimize this threat, we designed a search string that includes common terms for *"refactoring"* and *"bad smell"*. Furthermore, we performed a pilot search in each database to select a subset of the most common terms used in the research field, in order to retrieve the highest number of relevant papers. However, we may not assume that all existing related works were found by this filtering strategy.

Internal Validity: the guaranteed reproducibility of this study is due to the detailed specification of the search engines used, the search string, and the inclusion and exclusion criteria. Possible limitations of the search results were overcome by including different terms used by different authors, but that had similar concepts, staying in this SLR scope. Another threat may be related to the judgment of the information presented, which expresses only the authors point of view. To minimize this threat, the authors have carried out the selection stage, which in the initial stages made comparisons of the selected papers to avoid biasing in the selection of studies. Besides that, frequent meetings were held between all authors to discuss the relevant papers.

Conclusion Validity: the summarization of the data found in the literature and in the Fowler's catalog present the authors point of view, and may not present the actual concept conveyed by the papers. To minimize this threat, and to maintain the integrity of the information, we documented only what was explicitly presented by the papers.

External Validity: this threat is related to the representativeness of the selected papers published up to 2018 regarding the main goals of the systematic literature review. The systematic protocol was used to support a comprehensive representation of the selected papers, but some other papers may have been published after 2018 or indexed after the application of the search string in the databases. However, our findings about the relationship between refactoring and bad smells in the study period are accurate to the best of our knowledge.

5 Related Work

There are several reviews of the literature in the context of bad smells [15, 31, 34, 54] and refactoring [1, 29, 46]. However, most studies deal with these subjects individually, focusing just on one theme. Others deal with more than one theme, for instance, Sousa et al. [44] and Singh & Kaur [43].

Different from our work that focuses on the context of bad smells and refactoring, Sousa et al. [44] present a systematic literature mapping of studies that investigate the relationship between design patterns and bad smells. The authors focus on co-occurrence between design patterns and bad smells, providing a general analysis of the relationship between the GOF design patterns [14] and bad smells described by Fowler[18].

Singh & Kaur [43] perform a SLR of refactoring concerning code smells. Several data sets and tools for performing refactoring have been revealed and categorized depending on the detection approach: traditional method, visualization based technique, automatic method, semi-automatic method, empirical studies, and metric-based method. However, unlike this work, our studies treat these subjects together, arguing the explicit relationship between refactoring and bad smells discussed in the literature.

6 Conclusion

Refactoring from bad smell detection is not deeply discussed in the literature. This work presents the result of an SLR to identify the explicit relationship between refactoring and bad smells. 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 has also been found that: i) there are different refactoring strategies than those discussed by Fowler to address bad smells, ii) most strategies defined in the Fowler's book were addressed in the literature, and iii) most refactoring tools found may not detect bad smells.

As future work, we suggest an investigation on the feasibility of the refactoring strategies found in this SLR for the solution of bad smells proposed by Fowler, validating them and composing a more extensive and up-to-date catalog. Also, bad smells and refactorings strategies that were not proposed by Fowler and the new Fowler's catalog [17] should be investigated in comparison with the SLR findings. We also suggest an extension of RQ2, where research should be conducted focusing on identifying all refactoring tools present in the literature, that apply refactoring after detecting bad smells.

Acknowledgement

This research was partially supported by Brazilian funding agencies: CNPq (Grant 424340/2016-0), CAPES, and FAPEMIG (Grant PPM-00651-17)

References

1. Al Dallal, J.: Identifying refactoring opportunities in object-oriented code: A systematic literature review. IST **58**, 231–249 (2015)
2. bin Ali, N., Usman, M.: A critical appraisal tool for systematic literature reviews in software engineering. IST **112**, 48–50 (2019)
3. Azeem, M.I., Palomba, F., Shi, L., Wang, Q.: Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. IST **108**, 115–138 (2019)
4. Basit, W., Lodhi, F., Bhatti, U.: Extending refactoring guidelines to perform client and test code adaptation. In: Int. Conf. on Agile Soft. Development. pp. 1–13 (2010)
5. Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., Palomba, F.: An experimental investigation on the innate relationship between quality and refactoring. JSS **107**, 1–14 (2015)
6. Bavota, G., De Lucia, A., Marcus, A., Oliveto, R.: Recommending refactoring operations in large software systems. In: RSSE, pp. 387–419 (2014)
7. Bavota, G., Oliveto, R., Gethers, M., Poshyvanyk, D., De Lucia, A.: Methodbook: Recommending move method refactorings via relational topic models. TSE **40**(7), 671–694 (2013)
8. Beecham, S., Baddoo, N., Hall, T., Robinson, H., Sharp, H.: Motivation in software engineering: A systematic literature review. IST **50**(9-10), 860–878 (2008)
9. Boshnakoska, D., Mišev, A.: Correlation between object-oriented metrics and refactoring. In: Int. Conf. on ICT Innovations. pp. 226–235 (2010)
10. Cedrim, D., Garcia, A., Mongiovi, M., Gheyi, R., Sousa, L., de Mello, R., Fonseca, B., Ribeiro, M., Chávez, A.: Understanding the impact of refactoring on smells: A longitudinal study of 23 software projects. In: Proc. of the 11th JointMeeting on Foundations of Soft. Engineering. pp. 465–475 (2017)
11. Cedrim, D., Sousa, L., Garcia, A., Gheyi, R.: Does refactoring improve software structural quality? a longitudinal study of 25 projects. In: Proc. of the 30th SBSE. pp. 73–82 (2016)
12. Chatzigeorgiou, A., Manakos, A.: Investigating the evolution of code smells in object-oriented systems. Innovations in Syst. and Soft. Engineering **10**(1), 3–18 (2014)
13. Counsell, S., Hassoun, Y., Loizou, G., Najjar, R.: Common refactorings, a dependency graph and some code smells: an empirical study of java oss. In: Proc. of the ISESE. pp. 288–296 (2006)
14. Erich Gamma, Richard Helm, R.J., Vlissides, J.: Design patterns: Elements of reusable object-oriented software. Addison Wesley Longman Publishing (1994)
15. Fernandes, E., Oliveira, J., Vale, G., Paiva, T., Figueiredo, E.: A review-based comparative study of bad smell detection tools. In: Proc. of the 20th EASE. pp. 1–12 (2016)

16. Fontana, F.A., Zanoni, M., Zanoni, F.: A duplicated code refactoring advisor. In: Int. Conf. on Agile Soft. Development. pp. 3–14 (2015)
17. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional (2018)
18. Fowler, M., Beck, K., Brant, J., Opdyke, W.: Refactoring: Improving the Design of Existing Code. Addison-Wesley (1999)
19. Gupta, V., Kapur, P.K., Kumar, D.: Modelling and measuring code smells in enterprise applications using tism and two-way assessment. Int. Journal of Syst. Assurance Eng. and Management **7**(3), 332–340 (2016)
20. Haas, R., Hummel, B.: Learning to rank extract method refactoring suggestions for long methods. In: Int. Conf. on Soft. Quality. pp. 45–56 (2017)
21. Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S.: A systematic literature review on fault prediction performance in software engineering. TSE **38**(6), 1276–1304 (2011)
22. Higo, Y., Kamiya, T., Kusumoto, S., Inoue, K.: Refactoring support based on code clone analysis. In: Int. Conf. on Product Focused Soft. Proc. Improvement. pp. 220–233 (2004)
23. Kim, M., Zimmermann, T., Nagappan, N.: An empirical study of refactoring challenges and benefits at microsoft. TSE **40**(7), 633–649 (July 2014)
24. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. In Technical Report, Ver. 2.3 (EBSE) (2007)
25. Liu, H., Wu, Y., Liu, W., Liu, Q., Li, C.: Domino effect: Move more methods once a method is moved. In: 23rd SANER. vol. 1, pp. 1–12 (2016)
26. Liu, W., Hu, Z.g., Liu, H.t., Yang, L.: Automated pattern-directed refactoring for complex conditional statements. Journal of Central South University **21**(5), 1935–1945 (2014)
27. Liu, W., Liu, H.: Major motivations for extract method refactorings: analysis based on interviews and change histories. Front. of Com. Science **10**(4), 644–656 (2016)
28. Mahmoud, A., Niu, N.: Supporting requirements to code traceability through refactoring. RE **19**(3), 309–329 (2014)
29. Misbhauddin, M., Alshayeb, M.: Uml model refactoring: a systematic literature review. ESE **20**(1), 206–251 (2015)
30. Mkaouer, M.W., Kessentini, M., Bechikh, S., Cinnéide, M.Ó., Deb, K.: On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach. ESE **21**(6), 2503–2545 (2016)
31. Oliveira, J., Viggiato, M., Santos, M.F., Figueiredo, E., Marques-Neto, H.: An empirical study on the impact of android code smells on resource usage. In: SEKE. pp. 314–313 (2018)
32. Ouni, A., Kessentini, M., Sahraoui, H., Boukadoum, M.: Maintainability defects detection and correction: a multi-objective approach. ASE **20**(1), 47–79 (2013)
33. d. P. Sobrinho, E.V., De Lucia, A., d. A. Maia, M.: A systematic literature review on bad smells — 5 w's: which, when, what, who, where. TSE pp. 1–1 (2018)
34. Paiva, T., Damasceno, A., Figueiredo, E., Sant'Anna, C.: On the evaluation of code smells and detection tools. JSERD **5**(1), 7 (2017)
35. Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., De Lucia, A.: On the diffuseness and the impact on maintainability of code smells: A large scale empirical investigation. In: Proc. of the 40th ICSE. pp. 482–482 (2018)
36. Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., Poshyvanyk, D., De Lucia, A.: Mining version histories for detecting code smells. TSE **41**(5), 462–489 (2014)

37. Pietrzak, B., Walter, B.: Leveraging code smell detection with inter-smell relations. In: Int. Conf. on Extreme Programming and Agile Proc. in Soft. Engineering. pp. 75–84 (2006)
38. Rahman, M.M., Riyadh, R.R., Khaled, S.M., Satter, A., Rahman, M.R.: Mmruc3: A recommendation approach of move method refactoring using coupling, cohesion, and contextual similarity to enhance software design. SPE **48**(9), 1560–1587 (2018)
39. Sales, V., Terra, R., Miranda, L.F., Valente, M.T.: Recommending move method refactorings using dependency sets. In: 20th WCRE. pp. 232–241 (2013)
40. Sehgal, R., Mehrotra, D., Bala, M.: Analysis of code smell to quantify the refactoring. Int. Journal of Syst. Assurance Eng. and Management **8**(2), 1750–1761 (2017)
41. Silva, C., Santana, A., Figueiredo, E., Bigonha, M.A.S.: Revisiting the bad smell and refactoring relationship - data of the study, https://cleitonsilvat.github.io/eselaw2020/, acessed December 10, 2019
42. da Silva Carvalho, L.P., Novais, R.L., do Nascimento Salvador, L., de Mendonça Neto, M.G.: An approach for semantically-enriched recommendation of refactorings based on the incidence of code smells. In: ICEIS. pp. 313–335 (2017)
43. Singh, S., Kaur, S.: A systematic literature review: Refactoring for disclosing code smells in object oriented software. Ain Shams Engineering Journal (2018)
44. Sousa, B.L., Bigonha, M.A., Ferreira, K.A.: A systematic literature mapping on the relationship between design patterns and bad smells. In: Proc. of the 33rd Annual ACM SAC. pp. 1528–1535 (2018)
45. Tahir, A., Yamashita, A., Licorish, S., Dietrich, J., Counsell, S.: Can you tell me if it smells?: A study on how developers discuss code smells and anti-patterns in stack overflow. In: Proc. of the 22nd EASE. pp. 68–78 (2018)
46. Tavares, C.S., Ferreira, F., Figueiredo, E.: A systematic mapping of literature on software refactoring tools. In: Proc. of the XIV SBSI. p. 11 (2018)
47. Tsantalis, N., Chatzigeorgiou, A.: Identification of move method refactoring opportunities. TSE **35**(3), 347–367 (2009)
48. Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., Poshyvanyk, D.: When and why your code starts to smell bad. In: Proc. of the 37th ICSE. pp. 403–414 (2015)
49. Vale, G., Figueiredo, E., Abílio, R., Costa, H.: Bad smells in software product lines: A systematic review. In: Eighth SBCARS. pp. 84–94. IEEE (2014)
50. Vidal, S.A., Marcos, C., Díaz-Pace, J.A.: An approach to prioritize code smells for refactoring. ASE **23**(3), 501–532 (2016)
51. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proc. of the 18th EASE. p. 38 (2014)
52. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science & Business Media (2012)
53. Xia, X., Wan, Z., Kochhar, P.S., Lo, D.: How practitioners perceive coding proficiency. In: Proc. of the 41st ICSE. pp. 924–935 (2019)
54. Zhang, M., Hall, T., Baddoo, N.: Code bad smells: a review of current knowledge. Journal of Soft. Maint. and Evol.: Research and Practice **23**(3), 179–202 (2011)
55. Zhang, M., Hall, T., Baddoo, N., Wernick, P.: Do bad smells indicate" trouble" in code? In: Proc. of workshop on Defects in large software systems. pp. 43–44 (2008)