A Systematic Literature Mapping on the Relationship Between Design Patterns and Bad Smells

Bruno L. Sousa
Computer Science Department
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
bruno.luan.sousa@dcc.ufmg.br

Mariza A. S. Bigonha
Computer Science Department
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
mariza@dcc.ufm.br

Kecia A. M. Ferreira
Department of Computing
Federal Center for Technological Education of Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
kecia@decom.cefetmg.br

ABSTRACT

Bad Smells are symptoms that appear in the source code of a software system and may indicate a structural problem that requires code refactoring. Design patterns are solutions known as good practices that help building software systems with high quality and flexibility. Intuitively, it is possible to assume that the use of design patterns might avoid bad smells. Intriguingly, some recent studies have pointed out that this assumption is not true. This paper presents a systematic literature mapping of studies that investigate the relationship between design patterns and bad smells. We identified 16 papers which were categorized into three different approaches: impact on software quality, refactoring and co-occurrence. Amongst these three approaches, the co-occurrence relationship is the less explored in the literature. In addition, we identified that studies focusing on co-occurrence between design patterns and bad smells have generally analyzed the relationship between the GOF design patterns and bad smells described by Fowler and Beck. In this context, the Command design pattern was identified as the one with greater relationship with bad smells.

CCS CONCEPTS

• **General and reference** → **Empirical studies**; • **Software and its engineering** → **Design patterns**; • **Social and professional topics** → *Quality assurance*;

KEYWORDS

Design Pattern; Bad Smell; Systematic Literature Mapping

ACM Reference Format:
Bruno L. Sousa, Mariza A. S. Bigonha, and Kecia A. M. Ferreira. 2018. A Systematic Literature Mapping on the Relationship Between Design Patterns and Bad Smells. In *Proceedings of SAC 2018: Symposium on Applied Computing , Pau, France, April 9–13, 2018 (SAC 2018),* 8 pages.
https://doi.org/10.1145/3167132.3167295

1 INTRODUCTION

A Design pattern is a general solution applied to recurring problems in the context of software development [11], aiming to achieve a high reusability and extensibility. These solutions encourage the use of structures composed by inheritance, composition and polymorphism, in order to turn the communication between objects flexible, as well as to reduce the coupling between modules. In addition, design patterns establish a kind of communication between developers, facilitating the software architectural organization and comprehension. The Gang of Four's (GOF) book [11] describes 23 design patterns that are widely known and used by researchers and developers.

Bad smells are symptoms that exist in the structure or source code of a system and may indicate the presence of a more serious problem [10]. Code fragments with the presence of these symptoms are not considered errors, but, these structures may increase the complexity of the internal components of a software and negatively impact its architecture and comprehension. When these cases are identified, it is recommended to refactor the software system to eliminate potential problematic regions of the source code.

As design patterns and bad smells have opposite concepts, most literature studies have dealt with these two topics separately. However, other studies [6, 13, 14, 29] have identified the occurrence of structural relationships between them. Given the relevance of these concepts, the identification and analysis of studies that consider these two structures together is necessary in order to build a knowledge about the relationship between design patterns and bad smells.

This paper presents a Systematic Literature Mapping (SLM) regarding the relationship between design patterns and bad smells. This SLM has as main goals: (i) to provide an overview of the state of the art regarding to the relation between design patterns and bad smells; and (ii) to investigate if the co-occurrence relationship have been addressed by the literature. In the case papers addressing the co-occurrence relationship, we investigate: (i) which design patterns and bad smells are used to identify the co-occurrences; (ii) which co-occurrences were identified; and (iii) the methods used to identify co-occurrence in software. In an recent study, Cardoso and Figueiredo [5] mapped the relationships between design patterns and bad smells in two ways: structural and refactoring. The present study goes farther. Besides investigating both structural and refactoring relationship between design patterns and bad smells, we identified other relationships and detailed each of them.

The remainder of this paper is organized as follows. Section 2 describes the planning and execution of this systematic literature mapping, as well as discusses in details the process of searching and filtering the identified studies. Section 3 answers the research

SAC 2018, April 9–13, 2018, Pau, France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167295

questions of this study. Section 4 presents the main threats to validity of this systematic mapping. Section 5 concludes this study highlighting the main findings in this systematic mapping.

2 SYSTEMATIC LITERATURE MAPPING PROTOCOL

Systematic Literature Mapping (SLM) and Systematic Literature Review (SLR) are two important ways to aggregate and build knowledge in a specific area. For Kitchenham and Charters [16], systematic literature mapping is "a broad review of primary studies on a specific subject that seeks to identify the available evidence on a particular topic", while systematic literature review is defined as "a means of identifying, evaluating, and interpreting all available evidence relevant to a specific issue, thematic area, or phenomenon of interest". This paper presents a SLM, since we carry out a broad review on the topic of co-occurrence between design patterns and bad smells. The goals of an SLM are: (i) to provide a wide overview of a research area; (ii) to establish if research evidence exists on a topic; and (iii) to provide an indication of the quantity of evidence [16].

This SLM was performed in three phases: *planning*, *execution* and *analysis*. Section 2.1 presents the planning phase. Section 2.2 describes the execution, presenting the steps and the results of the selection process of the primary studies. Section 2.2.3 shows a list of papers obtained after the filtering steps that were analyzed in this SLM.

2.1 Planning

In the planning phase we defined: (i) the research to be investigated; (ii) the databases to search papers; (iii) the search string to be used; and (iv) the inclusion and exclusion criteria of the primary studies.

2.1.1 Research Questions. The research questions (RQn) aim to investigate and to understand how the literature has dealt with the relationship between design patterns and bad smells, more specifically the relation of co-occurrences between them.

Initially, we define two general purpose research questions.

RQ1: How has the literature addressed the relationship between design patterns and bad smells?

RQ2: Has the literature explored co-occurrence between design patterns and bad smells?

During the execution of the systematic mapping, we found studies that address co-occurrence between design patterns and bad smells. Thus, the RQ2 research question was subdivided into three other specific research questions, as follows.

RQ2.1: Which bad smells and design patterns are addressed by the literature for identifying co-occurrences?

RQ2.2: What co-occurrences have been identified by the studies?

RQ2.3: Which techniques have been used in the literature to find/establish the co-occurrence?

2.1.2 Electronic Databases. The electronic databases chosen for the collection of the primary studies are listed in Table 1. They were chosen because they are virtual libraries with a large collection of full works and metadata, recorded in `BibTex`, from published researches at conferences and journals of great importance to the academic community.

Table 1: Electronic databases.

Database	Address
ACM Digital Library	http://dl.acm.org/
Engineering Village	https://www.engineeringvillage.com
IEEE	http://ieeexplore.ieee.org/
Science Direct	http://www.sciencedirect.com/
Scopus	http://scopus.com/
Springer	http://link.springer.com/
Web of Science	http://webofknowledge.com/

2.1.3 Search String. To identify the relevant papers about design patterns and bad smells, we formulate a search string to find primary studies related to this mapping. Initially, the key words "design pattern" and "bad smell" were defined as the main terms of the expression. Next, a synonym search of these terms was carried out in order to refine this expression and to identify relevant and coherent studies with the proposed research questions. We defined the following search string.

("code smell" OR "code smells" OR "bad smell" OR "bad smells" OR "anti pattern" OR "antipatterns" OR "anti-pattern") AND ("design patterns" OR "design pattern")

2.1.4 Inclusion and Exclusion Criteria. The inclusion and exclusion criteria allow to classify each study in the mapping as a candidate to be included or excluded from the SLM, so that the papers may be restricted to the topic explored. As a systematic literature mapping may involve a large number of studies, we limited the scope of this SLM to select only complete papers, thus, we did not select studies classified as theses or dissertations. Table 2 shows the inclusion and exclusion criteria defined for this study.

Table 2: Inclusion and Exclusion Criteria.

Inclusion Criteria
Papers published in English.
Complete papers.
Papers published in Computer Science.
Papers available in electronic format.
Papers published in conferences and journals.
Papers related to the search string terms.
Exclusion Criteria
Documents classified as tutorials, posters, panels, lectures, round tables, theses, dissertations, book chapters and technical report.
Duplicate papers.
Papers that can not be located.

2.2 Execution

The execution phase, following the planning described in Section 2.1, consists in applying the search string in the databases to search primary studies. Next, the inclusion and exclusion criteria are applied for the studies selection.

2.2.1 Search Process. The electronic databases selected in this SLM have a kind of search engine that automates the search of papers. Thus, during the search of the primary studies, the search string was provided to each of these bases through a web graphic interface. The search process was carried out from January 15 to 20, 2017. During the search process, primary studies were not screened per year. We consider all studies returned by non-filter databases per publication year.

The final result of the first survey is shown in Table 3. We obtained 795 papers after the completion of this process.

Table 3: Studies obtained after the search process.

Data Base	Studies Returned
ACM Digital Library	12
Compendex (Engineering Village)	57
IEEE Xplore	0
Science Direct	176
Scopus	86
Springer	433
Web of Science	31
Total	**795**

The `IEEE Xplore` database did not return results. For this reason, studies referring to this database were not included.

2.2.2 Study Selection Process. After completing the step of searching for primary studies, we started a step of filtering these documents.

As there were a large number of papers identified in the search phase, the filtering process consisted of five steps. Each step proposed focus on the inclusion and exclusion criteria and relevance of the study according to its content. These steps are described following.

Step 1. It involves the elimination of duplicate studies. So, studies with same title and authors were discarded. This step eliminated 138 papers, resulting in an amount of 657 papers to be analyzed in Step 2.

Step 2. It consists in removing documents that are not papers. Thus, documents classified as tutorials, posters, panels, lectures, round tables, theses, dissertations, book chapters and technical report were removed at this step. This step eliminated 384 papers, resulting in an total of 273 paper to be analyzed in Step 3.

Step 3. The 273 titles and abstracts of the papers selected in the Step 2 were checked. Although these two structures aim to provide the reader with an idea about the content addressed, in some works it is not possible to understand, through them, if they are relevant or not for this SLM. For this reason, when analyzing some studies, there was some doubt about the selection or exclusion of them. Therefore, as a way to avoid hasty decisions, studies that presented this feature were classified as "doubtful" and passed on to Step 4 for further reading. This step identified 6 papers with relevance and 17 papers "doubtful". The "doubtful" papers were submitted to Step 4 for futher analysis.

Step 4. The 17 papers classified with "doubtful" in Step 3 were analyzed again. They passed for a diagonal reading. Diagonal reading is a reading of the introduction, topics and conclusion of a paper to find more details about it. At the end of this step, 5 out of 17 papers were identified relevant.

Step 5. At this point, we found 11 relevant papers for the SLM. However, searching in electronic databases does not guarantee that all main relevant studies on a particular topic will be retrieved. In order to minimize this limitation, a snowballing process was performed in this step. Snowballing is a search approach that uses citations in the papers from a systematic literature mapping as a reference list to identify other papers that have not yet been found [31]. There are two kind of snowballing that may be used to identify new studies: backward and forward. Backward snowballing means using the reference list of papers by the inclusion and exclusion criteria in order to identify new papers to include in the systematic mapping. Forward snowballing refers to identifying new papers based citing paper being examined. In this step, we adopted the backward snowballing strategy. Applying the backward snowballing approach, we analyzed 380 citations, referring to the 11 selected papers after Step 4, and five were choose to be part of the studies already selected.

In summary, Steps 1, 2, 3, 4 received as input, in this order, 795, 657, 273 and 17 primary studies; and Step 5 received 380 citations from the 11 studies already selected after Step 4. Figure 1 presents the results of each step, considering the primary studies received as input to each one. At the end we considered 16 papers in this SLM. They were analyzed and summarized in order to answer the research questions.

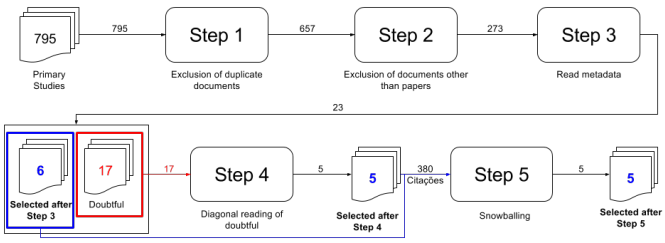

Figure 1: Filtering process conducted for selection of studies.

2.2.3 Results Summarization. We found 16 studies addressing the relationship between design patterns and bad smells. It is worthwhile to note that all selected papers are between the 2001 and 2017 years, and the most of them are concentrated from 2012 to 2017. This shows the topic has been discussed in literature recently.

The selected papers were read and summarized to extract the main information to answer the proposed research questions. We provided a summary of them in an attachment format. It is available on the research website[1].

[1] http://llp.dcc.ufmg.br/Publications/indexPublication.html

Table 4 lists the 16 primary studies selected and presents the main extracted data of them. They are sorted by year.

3 DISCUSSION OF RESULTS

This section presents the main results found in this SLM.

Analyzing the 16 studies of this SLM from Table 4, we identified three different categories of the relationship between design pattern and bad smell: *co-occurrence*, *impact on software quality* and *refactoring*.

The *co-occurrence* category consists in studies that investigated if design patterns and bad smells may occur together. The *impact on software quality* category consists in studies that evaluate in general the impact that the design patterns application exert on software quality. The *refactoring* category corresponds to studies that use design patterns as refactoring solutions for certain types of bad smells.

3.1 Relationship between Design Patterns and Bad Smells

This section answers the RQ1 research question.

RQ1: How has the literature addressed the relationship between design patterns and bad smells?

The co-occurrence category consists in studies that investigate if the design pattern components may present occurrence of bad smells or have static relationship with components with bad smells. From the data shown in Table 4, we identified that there are little studies that address this relationship. However, these studies are recent, being the first study published in 2013. In addition, to identify the co-occurrences, these studies have carried out case study with small to large open source systems software. The authors chose to evaluate co-occurrences through a case study because this method allowed the authors to carry out an exploratory study in order to identify the situations that impact on the emergence of the co-occurrences found.

In relation to the impact on software quality category some studies try to identify improvements in the design patterns application as well as to define new approaches which help the user to use correctly design pattern and prevent the bad smells emergence. In this category, as well as in the co-occurrences one, the studies indicate that the misuse of design patterns generate negative impacts on software quality. For instance, Izurieta and Bieman [12], McNatt and Bieman [19], and Wendorff [30] discuss some situations that may negatively impact software quality. According to them, design patterns application that do not fit the requirements and the design patterns coupling become the projects structure more complex, impairing some quality attributes such as: modularity, testability, and adaptability. Khomh and Gueheneuce [15] evaluated the quality of GOF design patterns via survey. This survey was based on the attributes: expansion capacity, simplicity, reuse, learning, comprehension, modularity, generalization, real-time modularity, scalability and robustness. Khomh and Gueheneuce [15] identified that design patterns do not always improve the software quality. They highlighted Flyweight as one of the design patterns that negatively affects software quality, decreasing the quality of all external software attributes except scalability. Vokac [27] investigated the GOF design patterns in order to identify possible defects which are ignored due to the good acceptance of these solutions. Vokac [27] analyzed a proprietary software and identified that Observer and Singleton design patterns were being used in complex areas with more code and a higher defect rate. In contrast, Factory Method design pattern presented the lowest defect rate. On the other hand Speicher [24] proposed an approach that considers decisions have taken by developers at the implementation time, and used them to identify bad smells and to improve software quality. Wagey et al. [28] proposed a quality model based on the use of design pattern to evaluate the external maintainability of a software. Via this model, the authors evaluated six open-source Java systems by a case study and came to the conclusion that the higher the pattern density in a software, the better its internal structure and the greater its maintainability, thus reducing part of the costs in the software maintenance phase.

In the refactoring category, the studies consider the premise that most of the time the choice of the design patterns is carried out manually, from the programmer's own knowledge. Due to this, a wrong choice may cause bad smells or generate a high complexity software structure. But if this choice is made and applied in an automated way, the design pattern may be applied correctly and provide the expected positive impacts. For instance, Christopoulou et al. [7] and Liu et al. [18] explored refactoring cases for the bad smell Complex Conditional Statements [10] using the Strategy design pattern as a refactoring solution. Liu et al. [18] also used the Abstract Factory pattern as another type of refactoring solution for this bad smell. Zafeiris et al. [33] considered the refactoring of Call Super [9], and used the Template Method design pattern as a solution. Nahar and Sakib [21, 22] have created a tool that analyzes UML class diagram and recommends refactoring with creational design patterns to the bad smells identified in the diagrams. Analyzing Table 4, we identified that in general, the authors validate the proposed approach via experiments. These studies have addressed only creational design patterns and two behavioral design patterns, Strategy and Template Method, for recommend refactoring solutions that remove bad smells of the source code. We did not identify refactoring proposals based on the others behavioral design patterns and structural design patterns. Thus, we can conclude regarding the refactoring category that besides the studies to support on the removal bad smell and improve the internal quality, it is a topic in which new studies may be extracted, such as proposals of refactoring strategies applying behavioral and structural design patterns, and creation of tools that suggest refactoring with design patterns for the removal of bad smells in a software.

Summary of RQ1. We conclude, in response to RQ1, that the literature has addressed the relationships between design patterns and bad smells in three different ways: co-occurrences, impact on software quality and refactoring. A great part of the studies have focused on the impact on software quality category, seven in total, and assessed the consequences and impacts of applying design patterns to software. The refactoring category presented the second largest number of studies, five in the total, and has proposed approaches and tools that apply design patterns to eliminate bad smells. The category of co-occurrences was the one that presented the smallest number of studies, four in total, and investigate relations of co-occurrence between design patterns, as well as the reasons that generated these relationships.

Table 4: Final result of the implementation phase of the Systematic Mapping.

Title	Author	Year	Relationship	#Systems	Type of System	System(s) Size	Method
1. Assessment of Design Patterns During Software Reengineering: Lessons Learned from a Large Commercial Project	Wendorff [30]	2001	Impact on software quality	1	Proprietary	1000 KLOC	Case Study
2. Coupling of Design Patterns: Common Practices and Their Benefits	McNatt and Bieman [19]	2001	Impact on software quality	-	-	-	Literature Review
3. Defect frequency and design patterns: An empirical study of industrial code	Vokac [27]	2004	Impact on software quality	1	Proprietary	24-32 KLOC	Case Study
4. Do Design Patterns Impact Software Quality Positively?	Khomh and Gueheneuce [15]	2008	Impact on software quality	-	-	-	Survey
5. Automated refactoring to the Strategy design pattern	Christopoulou et al. [7]	2012	Refactoring	8	Open and Proprietary	5-106 KLOC	Experiment
6. Analysing Anti-patterns Static Relationships with Design Patterns	Jaafar et al. [13]	2013	Co-occurrence	3	Open	182-241 KLOC	Case Study
7. A multiple case study of design pattern decay, grime, and rot in evolving software systems	Izurieta and Bieman [12]	2013	Impact on software quality	3	Open	43-119 KLOC	Case Study
8. Code Quality Cultivation	Speicher [24]	2013	Impact on software quality	1	Open	Not Mentioned	Case Study
9. Automated pattern-directed refactoring for complex conditional statements	Liu et al. [18]	2014	Refactoring	4	Open	Not Mentioned	Experiment
10. Automatic recommendation of software design patterns using anti-patterns in the design phase: A case study on abstract factory	Nahar and Sakib [21]	2015	Refactoring	6	Open	Not Mentioned	Case Study & Experiment
11. A proposal of software maintainability model using code smell measurement	Wagey et al. [28]	2015	Impact on software quality	6	Open	14-81 KLOC	Case Study
12. Co-Occurrence of Design Patterns and Bad Smells in Software Systems: An Exploratory Study	Cardoso and Figueiredo [6]	2015	Co-occurrence	5	Open	10-502 KLOC	Case Study
13. ACDPR: A Recommendation System for the Creational Design Patterns Using Anti-patterns	Nahar and Sakib [22]	2016	Refactoring	21	Open	Not Mentioned	Experiment
14. Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults	Jaafar et al. [14]	2016	Co-occurrence	3	Open	182-241 KLOC	Case Study
15. The relationship between design patterns and code smells: An exploratory study	Walter and Alkhaeir [29]	2016	Co-occurrence	2	Open	9-162 KLOC	Case Study
16. Automated refactoring of super-class method invocations to the Template Method design pattern	Zafeiris et al. [33]	2017	Refactoring	12	Open	18-180 KLOC	Experiment

3.2 Co-Occurrences between Design Patterns and Bad Smells

This section answers the RQ2 to RQ2.3 research questions.

RQ2: Has the literature explored co-occurrence between design patterns and bad smells?

The answer to this research question is yes. However, the results obtained in this SLM indicate that just a few studies have explored co-occurrences between design patterns and bad smells. Most studies have identified some co-occurrences that negatively impact the internal quality of software. Those co-occurrences have been attributed to the misuse of design patterns that increase the complexity of the software systems, or generate co-change relationships between classes.

Summary of RQ2. The result of this SLM suggests that co-occurrence between design patterns and bad smells has been discussed in the literature.

RQ2.1: Which bad smells and design patterns are addressed by the literature for identifying co-occurrences?

To answer this research question, Table 5 shows in details the design patterns and bad smells used in each paper classified in the co-occurrence category. Through Table 5 it is possible to observe that all the studies used GOF catalog design patterns. An important

factor that contributed for it was the existence of static source code analysis tools that automate the instances identification of software design patterns. The use of the tools provides greater agility in the information extraction process and enables a variety of design patterns to be detected efficiently and with low time and memory consumption. The tools used by these studies for detecting design patterns and bad smells in software are showed in Table 6.

We identified 18 bad smells used by these four papers. Analyzing them, we identified that these bad smells may be classified in three categories: bad smells proposed by Fowler and Beck [10], bad smells proposed by Brown et al. [4] and bad smells proposed by Lanza and Marinescu [17]. In addition, we identified that the bad smells proposed by Fowler and Beck [10] have been used in greater quantity followed by the bad smells proposed by Brown et al. [4] and by Lanza and Marinescu [17]. One explanation for this is the fact that there are a lot of tools that provide the collection of bad smells proposed by Fowler and Beck [10], in software systems.

Summary of RQ2.1. The Table 5 answer this research question. It shows that studies have used the GOF design patterns for co-occurrence identification in software systems. In addition, these works also have explored a large quantity of bad smells, a total of 18, that may be classified in three categories: bad smells proposed by Fowler and Beck [10], bad smells proposed by Brown et al. [4] and bad smells proposed by Lanza and Marinescu [17].

RQ2.2: What co-occurrences have been identified by the studies?

In the SLM performed, the Command design pattern was pointed out as the one presenting more co-occurrence with bad smell. In a case study performed by Cardoso and Figue-iredo [6], the authors identified the co-occurrence of Command with the God Class bad smells. By means of an exploratory analysis, the authors concluded that the excessive use of a simple receiver class in the application of the Command design pattern to different concerns caused the emergence of God Class bad smell. In addition, Cardoso and Figueiredo also found another kind of co-occurrence, Template Method with Duplicate Code. In this case, the multiple duplication of implementations of Template Method design pattern were responsible for its co-occurrence.

Jaafar et al. [13, 14] also identified the Command design pattern as the one with greater co-occurrence with bad smells among the design pattern analyzed, more precisely, Command with: Speculative Generality, Class Data Should Be Private, Long Method and Long Parameter List. Analyzing these relationships, the authors pointed out the following reason that led to emergence of these co-occurrence: (i) developers used public instance variables to allow the access of commands objects to data of others objects; (ii) commands object access features provided by others classes that perform lots of processing; and (iii) commands object may have relation with classes in the system that was engineered for a future extension, but they do not use them.

Finally, Walter and Alkhaeir [29] identified by a quantitative study the co-occurrence of Composite with Data Class and God Class, but they do not analyzed the reason that impacted on the emergence of these relationships in the systems analyzed. An interesting point to highlight is that all studies about co-occurrence relationship performed case study for identification of co-occurrences.

Summary of RQ2.2. The literature have been pointed out the Command design pattern as the one having more co-occurrence with bad smell. The studies analyzed in this SLM identified that it has relation with: God Class, Speculative Generality, Class Data Should Be Private, Long Method and Long Parameter List. In addition other three relationships were pointed out: Composite with God Class, Composite with Data Class and Template Method with Duplicate Code. These co-occurrences are summarized in Table 7.

RQ2.3: Which techniques have been used in the literature to find/establish the co-occurrence?

By analyzing available data in the studies, we observed that the co-occurrences were identified by the studies of two different ways. While Cardoso and Figueiredo [6] and Walter and Alkhaeir [29] used association rules [1, 3] to identify and analyze the co-occurrences, Jaafar et al. [13, 14] chose to use Fisher's exact test [23] to investigate these relationships.

Although these both methods may be used to examine associations between two items, they are applied and analyzed differently. Association rule is a method proposed in data mining that consists in combining items from a data set to extract knowledge about it. To identify the rules, it is necessary to compute some metrics: Support [1], Confidence [1], Lift [3] and Conviction [3]. Both Cardoso and Figueiredo [6] and Walter and Alkhaeir [29] used the Conviction metric as an analysis point of co-occurrences because it combines Support and Confidence into a single measure, showing how often an analyzed rule would be incorrect if the analyzed association could be attributed to a random chance.

Fisher' exact test is a statistical significance test used in the analysis of contingency tables that examine the significance of the associations between two classifications. Jaafar et al. [13, 14] used this test to check whether there is a significance between anti-pattern classes having static relationships with design patterns. To apply this method and analyze the results, the authors used a statistical environment called R[2].

Summary of RQ2.3. The studies have used association rules and Fisher' exact test to decide if exist co-occurrence between a design pattern and a bad smell. In the case of association rules, the authors have used the Conviction metric result to determine the co-occurrence existence. In the case of Fisher' exact test, the authors have used the R statistical environment to apply the method and evaluated the results significance to determine the co-occurrence existence.

4 THREATS TO VALIDITY

This section discusses some threats to the validity of this SLM and discusses some decisions to minimize them.

The search string of a SLM needs to be very well defined in order to return studies that are relevant to the search topic. In this study, several synonyms referring to the main terms of the SLM goal were searched. Some pilot searches were conducted in order to find new synonyms for the search string. Therefore, we believe that the defined search string has returned as many relevant papers as possible. However, it is not possible to state that all work concerning relations between design patterns and bad smells has been returned.

[2]https://www.r-project.org/

Table 5: List of design patterns and bad smells used by each study that address the co-occurrence relationship.

	Design Pattern	Bad Smell
Cardoso and Figueiredo [6]	Adapter, Command, Composite, Decorator, Factory Method, Observer, Prototype, Proxy, Singleton, Strategy, State, Template Method, and Visitor	God Class [17] and Duplicate Code [10]
Jaafar et al. [13]	Command, Composite, Decorator, Factory Method, Observer, and Prototype	Anti Singleton [4], Blob [4], Class Data Should Be Private [4], Complex Class [4], Long Method [10], Long Parameter List [10], Message Chain [10], Refused Parent Bequest [10], Spaghetti Code [4], Speculative Generality [10], and Swiss Army Knife [4]
Jaafar et al. [14]	Command, Composite, Decorator, Factory Method, Observer, and Prototype	Anti Singleton [4], Blob [4], Class Data Should Be Private [4], Complex Class [4], Long Method [10], Long Parameter List [10], Message Chain [10], Refused Parent Bequest [10], Spaghetti Code [4], Speculative Generality [10], and Swiss Army Knife [4]
Walter and Alkhaeir [29]	Adapter, Command, Composite, Decorator, Factory Method, Observer, Prototype, Proxy, Singleton, Strategy, State, Template Method, and Visitor	Data Class [10], Data Clumps [10], External Duplication [17], Feature Envy [10], God Class [17], Message Chains [10], and Schizophrenic Class [17]

Table 6: Tools used by the co-occurrence studies for detecting design patterns and bad smells in software.

Study	Design Pattern	Bad Smell
Cardoso and Figueiredo [6]	DPDSS [26]	JDeodorant [25] and PMD [8]
Jaafar et al. [13]	DeMIMA [2]	DECOR [20]
Jaafar et al. [14]	DeMIMA [2]	DECOR [20]
Walter and Alkhaeir [29]	DPDSS [26]	inCode [32]

Table 7: Summarization of the co-occurrences identified by the studies.

Study	Design Pattern	Bad Smell
Cardoso and Figueiredo [6]	Command Template Method	God Class Duplicate Code
Jaafar et al. [13, 14]	Command	Speculative Generality Class Data Should Be Private Long Method Long Parameter List
Walter and Alkhaeir [29]	Composite	Data Class God Class

The choice of electronic databases is another factor that may impact the results of a SLM. In this study, the primary studies were investigated in seven different electronic databases. However, other databases that were not used in the survey may contain work that are relevant to this mapping. To mitigate this threat, a step that consists in a snowballing process [31] was performed during the filtering phase of the papers. In this step, the citations of the selected papers were verified through a list of references in order to find other relevant studies that had not been returned.

This SLM considered only papers written in the English. It is possible that some relevant studies may be written in other languages. However, the main means of scientific publication in Software Engineering accepts papers in English. Therefore we consider that using English is sufficient to filter the main studies in the subject.

Finally, the data extraction referring to the selected papers was carried out only by the main author of this paper. This may be considered a threat to validity, since such analysis was performed subjectively and based on the knowledge of only one of the authors. However, the papers were discussed by the three authors. In addition, the classification scheme is another point that is considered a threat to validity, since it was also performed subjectively from the extraction of the data. However, these categories have been proposed only as a grouping guide of the studies with common focus to facilitate the reading and the understanding of the readers.

5 CONCLUSION

This paper presents a SLM in order to provide an overview of the state of the art with respect to the relation between design patterns and bad smells, specifically analyzing the relations of co-occurrence between these two structures.

In this SLM, we analyzed 16 papers. The results of this SLM show that the literature has investigated relationships between design patterns and bad smells in three different ways: co-occurrence, impact on software quality and refactorig. The analysis carried out in these studies indicate that when the design pattern implementation causes bad smells, consequently it degrades software quality, increasing complexity and damaging other important external attributes such as modularity, flexibility, testability, among others. However, when the design pattern application is well-designed, as described in the studies on refactoring relationships [7, 18, 21, 22, 33], the impacts generated are positive.

In addition, these results indicate that the co-occurrence relationship between design patterns and bad smells is a current topic, and has been explored since 2013. The bad smells described by Fowler and Beck [10] are the most used in these investigations. However, these studies have also addressed some of the bad smells described

by Brown et al. [4] and Lanza and Marinescu [17]. Regarding design patterns, the studies have addressed solutions integrating the GOF catalog, proposed by Gamma et al. [11]. The use of these design patterns is justified by the existence of tools that perform the extraction of the instances automatically.

Finally, the literature have been pointed out the Command design pattern as the one having more co-occurrence with bad smell. The studies identified that this design pattern has co-occurrence with: God Class, Speculative Generality, Class Data Should Be Private, Long Method and Long Parameter List. In addition, other three relationships were pointed out: Composite with God Class and Data Class, and Template Method with Duplicate Code. The authors identified the main situations that contributed for the emergence of these co-occurrences: the misuse or inappropriate application of certain design pattern, the misuse planning of a system, and excessive assignment of functionality to the design patterns internal components. With respect to method for identification of co-occurrence, we identified that the authors have used two ways: association rules and Fisher' exact test. In the case of association rules, the authors have used the Conviction metric as a analysis point to determine the co-occurrence existence. In the case of Fisher' exact test, the authors have used the R statistical environment to apply the method and evaluated the significance of the test to determine the co-occurrence existence.

As future work we intend to (i) investigate co-occurrence relationships with other design patterns and bad smells not explored by the studies contained in the co-occurrence category; (ii) investigate the design patterns, bad smells, and software failures relationships; and (iii) build a refactoring recommendation system that helps to eliminate bad smells from the source code of a software system.

6 ACKNOWLEDGE

This work was sponsored by CAPES, CNPQ and FAPEMIG.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules Between Sets of Items in Large Databases. *SIGMOD Rec.* 22 (1993), 207–216.

[2] Giuliano Antoniol and Yann-Gaël Guéhéneuc. 2008. DeMIMA: A Multilayered Approach for Design Pattern Identification. *IEEE Transactions on Software Engineering* 34 (2008), 667–684.

[3] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. 1997. Dynamic Itemset Counting and Implication Rules for Market Basket Data. *SIGMOD Rec.* 26 (1997), 255–264.

[4] William H. Brown, Raphael C. Malveau, Hays W. "Skip" McCormick, and Thomas J. Mowbray. 1998. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis* (1st ed.). John Wiley & Sons, Inc., New York, USA.

[5] Bruno Cardoso and Eduardo Figueiredo. 2014. Co-Occurrence of Design Patterns and Bad Smells in Software Systems: A Systematic Literature Review. In *Proceedings of the 11th Workshop on Software Modularity*. 82–93.

[6] Bruno Cardoso and Eduardo Figueiredo. 2015. Co-Occurrence of Design Patterns and Bad Smells in Software Systems: An Exploratory Study. In *11st SBSI*. 347–354.

[7] Aikaterini Christopoulou, E. A. Giakoumakis, Vassilis E. Zafeiris, and Vasiliki Soukara. 2012. Automated refactoring to the Strategy design pattern. *Information and Software Technology* 54, 11 (2012), 1202–1214.

[8] F. A. Fontana, M. Zanoni, A. Marino, and M. V. Mäntylä. 2013. Code Smell Detection: Towards a Machine Learning-Based Approach. In *2013 IEEE International Conference on Software Maintenance*. 396–399.

[9] Martin Fowler. 2015. CallSuper. https://martinfowler.com/bliki/CallSuper.html. (2015). Accessed March 2017.

[10] M. Fowler and K. Beck. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

[11] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing.

[12] Clemente Izurieta and JamesM. Bieman. 2013. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal* 21, 2 (2013), 289–323.

[13] Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Foutse Khomh. 2013. Analysing Anti-patterns Static Relationships with Design Patterns. *Electronic Communications of the EASST* 59 (2013).

[14] Fehmi Jaafar, Yann Gueheneuc, Sylvie Hamel, Foutse Khomh, and Mohammad Zulkernine. 2016. Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults. *Empirical Software Engineering* (2016), 896–931.

[15] Foutse Khomh and Yann-Gael Gueheneuce. 2008. Do Design Patterns Impact Software Quality Positively?. In *Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*. 274–278.

[16] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE 2007-001. Keele University and Durham University Joint Report. http://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf

[17] Michele Lanza and Radu Marinescu. 2006. *Object-Oriented Metrics in Practice*. Springer-Verlag N. Y.

[18] W.a Liu, Z.-G.a b Hu, H.-T.b Liu, and L.b Yang. 2014. Automated pattern-directed refactoring for complex conditional statements. *Journal of Central South University* 21, 5 (2014), 1935–1945.

[19] William B. McNatt and James M. Bieman. 2001. Coupling of Design Patterns: Common Practices and Their Benefits. In *Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development*. 574–579.

[20] Naouel Moha, Yann-Gael Gueheneuc, Laurence Duchien, and Anne-Francoise Le Meur. 2010. DECOR: A Method for the Specification and Detection of Code and Design Smells. *IEEE Trans. Softw. Eng.* 36, 1 (Jan. 2010), 20–36.

[21] Nadia Nahar and Kazi Sakib. 2015. Automatic recommendation of software design patterns using anti-patterns in the design phase: A case study on abstract factory. In *CEUR Workshop Proc.* 9–16.

[22] Nadia Nahar and Kazi Sakib. 2016. ACDPR: A Recommendation System for the Creational Design Patterns Using Anti-patterns. In *IEEE 23rd SANER*. 4–7.

[23] David J. Sheskin. 2007. *Handbook of Parametric and Nonparametric Statistical Procedures* (4 ed.). Chapman & Hall/CRC.

[24] D. Speicher. 2013. Code Quality Cultivation. *Communications in Computer and Information Science* 348 (2013), 334–349.

[25] Nikolaos Tsantalis, Theodoros Chaikalis, and Alexander Chatzigeorgiou. 2008. JDeodorant: Identification and Removal of Type-Checking Bad Smells. In *Proc. of the 12th CSMR*. 329–331.

[26] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T Halkidis. 2006. Design pattern detection using similarity scoring. *Software Engineering, IEEE Transactions on* 32, 11 (2006), 896–909.

[27] Marek Vokac. 2004. Defect frequency and design patterns: An empirical study of industrial code. *IEEE Transactions on Software Engineering* 30, 12 (2004), 904–917.

[28] B. C. Wagey, B. Hendradjaya, and M. S. Mardiyanto. 2015. A proposal of software maintainability model using co- de smell measurement. In *ICoDSE*. 25–30.

[29] Bartosz Walter and Tarek Alkhaeir. 2016. The relationship between design patterns and code smells: An exploratory study. *Information and Software Technology* (2016), 127–142.

[30] Peter Wendorff. 2001. Assessment of Design Patterns During Software Reengineering: Lessons Learned from a Large Commercial Project. In *5th CSMR*. 77–84.

[31] Claes Wohlin. 2014. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *18th EASE*. 1–10.

[32] Aiko Yamashita and Leon Moonen. 2013. To What Extent Can Maintenance Problems Be Predicted by Code Smell Detection? - An Empirical Study. *Inf. Softw. Technol.* 55, 12 (Dec. 2013), 2223–2242.

[33] Vassilis E. Zafeiris, Sotiris H. Poulias, N.A. Diamantidis, and E.A. Giakoumakis. 2017. Automated refactoring of super-class method invocations to the Template Method design pattern. *Information and Software Technology* (2017), 19–35.