On the Investigation of Domain-Sensitive Bad Smells in Information Systems

Markos Viggiato, Johnatan Oliveira, Eduardo Figueiredo
Software Engineering Laboratory (LabSoft)
Department of Computer Science
Federal University of Minas Gerais (UFMG)
Belo Horizonte, Brazil
{markosviggiato, johnatan.si, figueiredo}@dcc.ufmg.br

ABSTRACT

Bad smells are symptoms that something may be wrong in the information system design or source code. Although bad smells have been widely studied, we still lack an in-deep analysis about how they appear more or less frequently in specific information systems domains. The frequency of bad smells in a domain of information systems can be useful, for instance, to allow software developers to focus on the more relevant bad smells of a certain domain. Moreover, developers of new bad smell detection tools could take information about domains into consideration to improve the tool detection rates. In this paper, we investigate code smells more likely to appear in four specific information systems domains: accounting, e-commerce, health, and restaurant. Our analysis relies on 52 information systems mined from GitHub. We identified bad smells with two detection tools, PMD and JDeodorant. Our findings suggest that Comments is a domain-independent bad smell since they uniformly appear in all investigated domains. On the other hand, Large Class and Long Method can be considered domain-sensitive bad smells since they appear more frequently in accounting systems. Although less frequent in general, Long Parameter List and Switch Statements also appear more in health and e-commerce systems, respectively, than in other domains.

CCS Concepts

•**Software and its engineering** → **Contextual software domains; Software maintenance tools; Software design engineering; Software defect analysis; Software evolution; Maintaining software;**

Keywords

Bad Smell, Information System Domain, Detection Tools.

SBSI 2017 June 5^{th} – 8^{th}, 2017, Lavras, Minas Gerais, Brazil

1. INTRODUCTION

A bad smell is any symptom that may indicate a deeper quality problem in the information system design or source code [7]. Bad smells are considered expensive because they represent poor solutions that hinder software maintenance tasks [5]. Several factors may contribute to the addition of bad smells in information systems. For instance, software developers can introduce a bad smell due to their wrong understanding of the system context, including misunderstanding of domain-specific requirements. In fact, previous works suggest that software quality and the presence of bad smells may depend on the information system domain [3, 5]. However, code smell detection tools ignore information related to the system domain [8].

Many studies have been published in the literature on code smells and their detection strategies [1, 4, 18]. However, the relationships between bad smells and information systems domains have been little studied so far. Most research that investigates bad smells in systems domains reports only on preliminary small-scale studies [3, 5, 8]. In addition, they found conflicting results. For instance, Fontana et al. [5] identified that Duplicate Code, Data Class, Large Class, and Long Method are in general most common, but at the domain level, significant differences among bad smells were not observable. On the other hand, Linares-Vasquez [13] found that some bad smells are common in all domains while others, such as Blob (see in [15]), are more common in certain domains (e.g., Science and Education).

It is important to know how frequent a bad smell is in a domain for several reasons. For instance, this information could help developers of information systems to focus their attention on the smells that mostly contribute to the deterioration of the source code quality. It can also help the development of new bad smell detection tools with better detection rates since different domains may require different detection strategies. Therefore, if the frequency of bad smells significantly varies by domains, then it is important to report such variations and to understand why they occur.

This paper describes an empirical study on the detection of bad smells, aiming at identifying the most frequent smells in different domains of information systems domains. We perform an analysis on 52 object-oriented Java information systems mined from GitHub of 4 different domains: accounting, e-commerce, health, and restaurant. We rely on PMD [5] and JDeodorant [22] to detect 6 types of bad smells in the target information systems. However, we discarded JDeodorant tool, because we were not able to execute it in

information systems that are not buildable in Eclipse. As there are several systems in this condition, we performed our study only using the PMD tool. The detected bad smells are Large Class, Long Method, Long Parameter List, Switch Statements, Comments, and Dead Code.

From our findings, we can classify the bad smells in 2 groups, namely: (i) domain-sensitive bad smells that appear more frequently in some domains than in others and (ii) domain-independent bad smells that appear in all domains with no significant difference. For instance, Comments is a domain-independent bad smell since it uniformly appears in all investigated domains. On the other hand, Large Class and Long Method can be considered domain-sensitive bad smells since they appear more frequently in accounting systems and the later is also very common in health systems. In this study, we also observe that Long Parameter List and Switch Statements are rare in all domains. Although less frequent, Long Parameter List appear more in health systems while Switch Statements is more common in e-commerce systems.

The remainder of this paper is organized as follow. Section 2 presents a background to support the comprehension of our work. Section 3 presents the configurations of our study. Section 4 reports and discusses the obtained results. Section 5 indicates the threats to the study validity. Section 6 discusses related work. Finally, Section 7 concludes our study and discusses some points for further work.

2. BACKGROUND

In this section we define the bad smells used in this study and the strategies for detecting bad smells.

2.1 Bad Smells

Bad smells are symptoms that may indicate a deeper quality problem in the information system design or code [7]. A bad smell may have been caused by poor design choices or by misunderstanding of domain-specific requirements [7, 9, 19, 20]. Bad smells are often associated with increasing in development and maintenance costs since it is harder for software developers to modify and evolve an information system that contains many bad smells [19]. Fowler [7] defined a set of 22 bad smells and we selected 5 among them, because previous studies have shown that they are very common in information systems. We also included Dead Code in this study since it is one of the most studied and used bad smells [2]. Table 1 presents a brief definition of each bad smell considered in this study. The definitions are in accordance with Fowler [7] and Lanza et al. [12].

During the process of choice of these bad smells we also took in account that they can be automatically detected by the tools we used: PMD and JDeodorant. In total, our analysis relies on 6 types of bad smells: Large Class, Long Method, Long Parameter List, Switch Statements, Comments, and Dead Code.

2.2 Detection Strategies and Tools

There are several techniques to identify bad smells [1, 11, 16]. Bad smells can be detected in source code by either using manual or automated analysis. Tools support automated analysis relying usually on different detection strategies, such as software metrics [12, 16] and program slicing [10]. This variety of strategies allows detection of different types of bad smells. However, it is important to highlight that, as far as we are concerned, existing bad smell detection tools do not use information related to domain of the information systems [5]. In this paper, we used 2 bad smell detection tools: PMD and JDeodorant. Their characteristics can be verified in Table 2 and are detailed as follows.

Table 1: Types of bad smells

Bad Smell	Definition
Large Class	It defines a class that tends to centralize the intelligence of the system, for instance, with several methods and attributes. It usually has an excessive code size.
Long Method	It is a method too long in Lines of Code so it is difficult to be understood and changed. In general, it tends to centralize the functionality of a class, similarly to a Large Class.
Comments	It occurs when large blocks of comments, written to explain poorly implemented code snippets
Dead Code	Code that has been used in the past, but is not currently used
Long Parameter List	It occurs when the parameter list in a method is too long and thus difficult to understand.
Switch Statements	Identified when the same switch statement (or âĂIJifâĂęelseâĂİ, statement) is scattered in a program in many different places.

Table 2: Bad smells detection tools

Features	JDeodorant	PMD
Type	Eclipse plug-in	Eclipse plug-in and Standalone
Version	5.0.64 / 2016	5.5.4 / 2017
Supported Languages	Java	C,C#, C++, JAVA PHP and 11 others
Bad Smell Detected	Large Class Feature Envy Long Method Type Checking	Dead Code Comments Large Class Long Method Duplicated Code Long Parameter List

PMD is an open-source tool for Java and an Eclipse plug-in. It searches for potential issues in the source code, such as Dead code, empty try/catch blocks, Long Switch Statements, Unused Local Variables, and Over Complicated Expressions. Moreover, PMD allows the user to set parameters to customize its detection strategies [6, 14, 17]. In our study, however, we rely on the default configuration of both tools.

JDeodorant is an open-source tool that automatically identifies 4 bad smells [22]: Large Class, Large Method, Feature Envy, and Switch Statements. JDeodorant uses metrics and program slicing techniques to detect bad smells [22]. PMD and JDeodorant were selected because are available for download and are free for use. Besides, both tools have been actively developed and maintained [6]. Other studies on bad smells have also used these tools [1, 4, 24]. Table 2 presents information about the selected tools.

3. STUDY SETTINGS

This section describes an empirical evaluation to identify domain-sensitive bad smells. For this purpose, we designed an exploratory study conducted based on guidelines for empirical studies [23]. Section 3.1 presents the study goal and research questions designed to guide our study. Section 3.2 describes the phases to evaluate our study. Finally, Section 3.3 discusses the steps for collecting the target information systems from GitHub.

3.1 Goals and Research Questions

The main goal of this study is to analyze the occurrence of bad smells in information systems from the following domains: accounting, e-commerce, health, and restaurant. We are interested in assessing the domain-sensitive bad smells and the domain-independent ones. For this purpose, we conceived the following research questions (RQs) to guide our study.

RQ1 *What are the most frequent bad smells in each information system domain?*

RQ2 *What are the domain-independent bad smells?*

Through RQ1, we are interested on investigating the feasibility to identify and list the bad smells that are more common within the selected domains. On the other hand, with RQ2, we aim to identify and catalog bad smells that uniformly appear regardless the software domain, i.e., those bad smells that have high chance to occur in many application domains with no significant difference.

3.2 Evaluation Phases

To answer the research questions presented in Section 3.1, we designed a study composed of four phases presented in Figure 1. Each phase is discussed as follows.

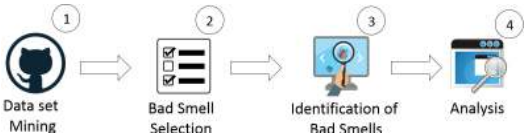

Figure 1: Study Phases

Phase 1) Data Set Mining - We searched for information systems sorted by stars in GitHub. Stars are a meaningful measure for repository popularity among the platform users, and they may be used to support the selection of well evaluated projects. To retrieve the information systems, we used some keywords such as: accounting, e-commerce, electronic commerce, hospital, infirmary, and restaurant. Section 3.3 presents the data set extraction in more details.

Phase 2) Bad Smell Selection - We selected 6 types of bad smells: Large Class, Long Method, Long Parameter List, Switch Statements, Comments, and Dead Code. This selection was adopted because previous studies have shown that they are very common in information systems and the results obtained by these studies are conflicting [5, 13]. There is, also, a lack of an in-deep analysis of these smells according to the system domains, trying to establish a relationship between them. In addition, these bad smells can be automatically detected by the tools we used: PMD and JDeodorant. It is important to note that we discarded JDeodorant tool, because we were not able to execute it in information systems that cannot be built in Eclipse. As there are several systems from GitHub in this condition, we performed our study using only PMD detection tool.

Phase 3) Identification of bad smells - To avoid biasing the results, we decided to use both tools in the default configuration. We analyzed each information system at a time in Eclipse, using the plug-in of the selected tools.

Phase 4) Analysis - We cloned the information systems from the four domains from GitHub. We run the two tools and compute their output results. Each bad smell from each domain was stored in spreadsheets so that we could calculate the occurrences by domain, with the aim of identifying the more frequent bad smells (domain-sensitive) and the domain-independent ones.

3.3 Data Set Mining

To evaluate our study, we chose systems from the mentioned domains for several reasons. First, these information systems, in general, are intuitive systems and easy to evaluate. Second, there is a large number of these systems available for download in GitHub. Third, since the selected systems are within well-defined domains, we believed that it would be easy to identify domain-specific requirements that influence the number of bad smells. The systems that compose our data set were retrieved from GitHub repositories in October 2016. We performed 5 steps to collect the information systems from GitHub, as indicated in Figure 2 and described as follows.

Figure 2: Steps for Collecting Systems from GitHub

In step (1), we performed a preliminary search to evaluate the feasibility of collection of the selected information systems. In step (2), we define appropriate search strings per domain since there is a diverse terminology to represent the same software domain on GitHub. For instance, we may refer to the e-commerce domain as ecommerce, without hyphenation. Thus, to collect the information systems that compose our data set we developed an algorithm to search automatically within GitHub. Since the goal of our study is to detect bad smells from different information systems, given large system sets per domain, we defined the search strings presented in Table 3.

Table 3: Search string per domain

Domain	Search String
Accounting	Accountancy OR Accounting
Restaurant	Restaurant OR Eatery OR Restaurants
Health	Hospital OR Infirmary OR Health
E-Commerce	E-Commerce OR Ecommerce OR Electronic Commerce

In step (3), we run the algorithm, as mentioned in the step (2) to clone the information systems to a local storage.

This step is necessary, because we know that several systems are hosted in GitHub and a manual cloning would be infeasible. From the previous steps, it was possible to obtain 400 information systems, 100 for each domain sorted in descending order by stars. In step (4), aiming at restricting our data set in order to obtain the most relevant systems, we applied the following exclusion criteria: First, non-Java information systems (remaining 280 systems), since GitHub does not verify automatically the main programming languages of the systems and the selected tools are specific to Java programming languages. Second, projects developed for Android platform (remaining 130 systems), because Android systems tend to have a different architectural design and code implementation when compared with traditional Java systems. Third, systems with less than 1,000 lines of code (remaining 52 systems). After applying these filters presented in step (4), we obtained 52 systems. Therefore, in step (5), our data set is finally composed by 52 information systems to support the identification of the bad smells.

To better characterize the our data set, we computed the metrics from Metrics[1] plug-in. Table 4 presents the following metrics computed per domain: lines of code (LOC), number of classes (NOC), number of methods (NOM) and number of attributes (NOA), respectively. The presented values correspond to the sum of each metric for the respective domain. As we can see, the accounting domain is the biggest one in code size, having more than 71 KLOC, while restaurant is the smallest one, with less than 32 KLOC.

Table 4: Data Set Characterization

Software Domains	# Systems	LOC	NOC	NOM	NOA
Accounting	10	71,547	387	5,619	4,367
E-commerce	19	57,366	832	5,531	2,277
Hospital	11	49,121	392	3,557	2,957
Restaurant	12	31,872	468	3,183	1,362

4. RESULTS

This section presents the results and analysis. We analyzed 52 systems mined from GitHub from accounting, e-commerce, health, and restaurant domains. Six bad smells were analyzed in these systems with the PMD detection tool. We organize our discussion in four parts. Section 4.1 presents an overview of the detected bad smells in our data set. Section 4.2 discusses the percentage of systems with bad smells found in each domain. Section 4.3 analyzes the percentage of occurrence of each bad smell according to the entity which the smell is related to. For example, the percentage of Large class is related to the total number of classes within the domain, the percentage of Long method and Long parameter list are evaluated according to the number of methods in the domain since these bad smells are related to methods, and so on. Finally, Section 4.4 presents a joint frequency analysis by systems and by entities.

4.1 Overview

In order to understand how the bad smells are distributed over our dataset, we first investigated all systems from all domains together. Figure 3 shows the percentage of systems with each bad smell in all domains. Data in this figure show that four (out of six) kinds of bad smells are very common in the selected systems. In fact, we detected Comments in all systems of our data set.

According to our data, the second more common bad smell is Dead Code. This bad smell was detected in 73% of the systems in all domains. Large Class and Long Method are also very common since they could be found in about 60% of the systems. On the other hand, Long Parameter List and Switch Statements are the least frequent bad smells, appearing, respectively, in 23% and 13% of the systems.

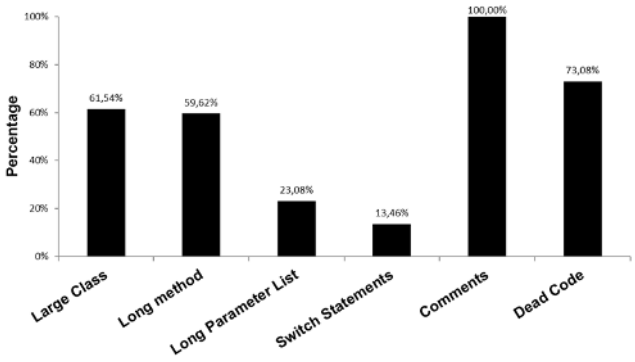

Figure 3: Percentage of Systems with Bad Smells

4.2 Frequency Analysis by Systems

With the aim of identifying which bad smells are more common in each domain, we conducted a detailed study on the frequency of each selected bad smell in systems of specific domains. Figure 4 presents the six bad smells and their frequency in each domain. For instance, Comments was found in all systems of every domain. Therefore, its frequency is 100% for all domains. This result is expected since all systems in our data set have Comments, as discussed in Section 4.1. Although highly frequent, Comments may not be considered a serious problem because someone could argue that they do not directly affect the system behavior. However, according to Fowler [7], Comments may be used to hide a possible bad design and that is why we decided to include it in our analysis.

Apart from Comments, Large Class and Long Method are the most frequent bad smells in accounting systems. In fact, all accounting systems in our data set have at least one instance of Large Class. Similarly, Long Method is in about 90% of systems in this domain. It is interesting to see, however, that these two bad smells are not so common in the other domains. That is, Large Class could only be found in about 50% of systems in the e-commerce, health, and restaurant domains. Long Method was found in more than 60% of e-commerce systems, the second domain with higher frequency. However, it is still clear that both Large Class and Long Method seem to be domain-sensitive bad smells with the highest frequency in accounting systems.

Figure 4 also shows interesting results for Long Parameter List. PMD found this bad smell in about 45% of systems in the heath domain. On the other hand, only 25% of e-commence systems and 20% of accounting systems have Long Parameter List. This large difference suggests that Long Parameter List is a domain-sensitive bad smell

[1]http://metrics.sourceforge.net/

with the highest frequency in health systems. Our data do not allow us to say whether Dead Code and Switch Statements are domain-independent or domain-sensitive bad smells.

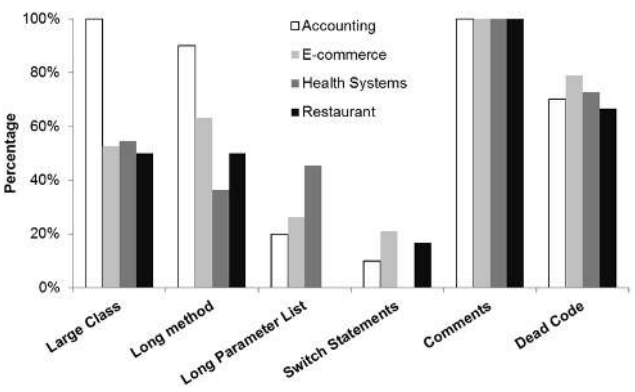

Figure 4: Frequency of Bad Smells in Software Domains

4.3 Frequency Analysis by Entity

The previous section presented the percentage of systems with a bad smell in each domain. However, a bad smell may appear in most systems of a domain, but in only a few parts of these systems. That is, it could be rare in classes of a system, although some instances exist in most systems.

To address this point, this section analyzes the frequency of occurrence of each bad smell with respect to the entity with which the smell is related to. For instance, if we are analyzing the frequency of Large Class, we have to divide the number of smells by the number of classes. The information about the frequency by entity and the frequency by systems complement each other to support stronger conclusions about domain-sensitive bad smells.

The entity is defined according to each bad smell. In this way, we have the frequency of Large Class evaluated in relation to the number of classes (NOC), which is our first entity. Since the smells Long Method and Long Parameter List are related to the method entity, their frequency is calculated according to the number of methods (NOM) within the domain. Finally, Switch Statement and Dead Code have no relation with a specific entity and then we obtain their frequency of occurrence using KLOC. It is important to note that we excluded Comments from this analysis because we observed that it is highly frequent in all domains (Section 4.2).

In order to evaluate the frequency of Large Class related to the class entity for each domain, we first count the total number of Large Class in a domain and then we divide this number by the total number of classes. This procedure is done for every domain separately and the results are presented in Figure 5.

Data in Figure 5 show that Large Class has the highest frequency of occurrence in the accounting domain in comparison to other domains. In fact, it is present in approximately 0.18 (18%) of the classes of accounting, i.e., 18% of the classes of this domain are Large classes. E-commerce and health have about 4% of Large classes each one, while restaurant has the lowest frequency, approximately 3%.The

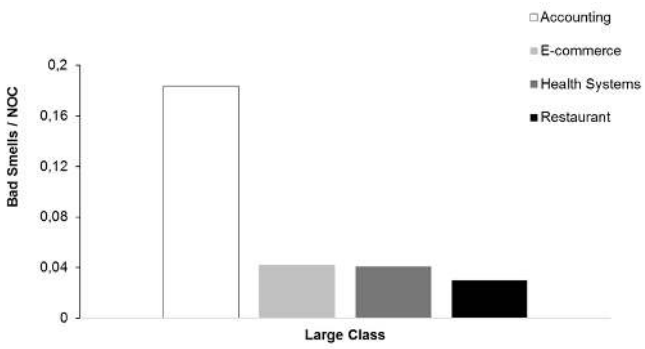

Figure 5: Frequency of Large Class in Domains by Entity

significant difference between accounting and the other domains suggests that Large Class is a domain-sensitive bad smell appearing more frequently in the accounting domain.

We believe that the high frequency of Large Class in the accounting domain is due the number of calculations done, what makes the class to have an excessive code size. In fact,by looking at the source code, we identified a high number of calculations within a single class, such as different kinds of fees and charges.

As for Large class, the same procedure was done for Long Method and Long Parameter List. However, we have the method as the entity now. In Figure 6, we can observe that long methods occur more frequently in accounting domain being present in almost 1.8% of all methods. Next, we have health and e-commerce domains having a frequency of 1.2% and 0.9%, respectively. Restaurant has the lowest value, about 0.6% of occurrence of long methods among the methods of the systems. These numbers may indicate that Long Method is a domain-sensitive bad smell since it appears more frequently in accounting and health system than in others, what is in line with the frequency analysis by systems presented in Section 4.2.

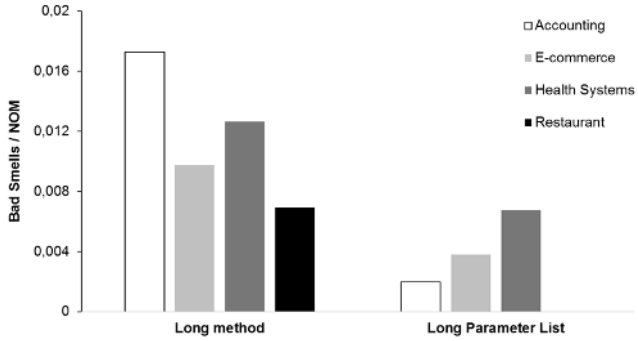

Figure 6: Frequency of Long Method and Long Parameter List in Domains by Entity

Long Methods tend to be common in accounting systems since, as we have already seen, Large Classes are also very common due to the great number of calculations. These calculations are done within the methods, what makes them very large in code size and also makes them to concentrate the behavior of the class.

Long Parameter List occurs more frequently in health systems with a percentage of 0.6%, followed by e-commerce and accounting. We can note that the restaurant domain does not have any occurrence of Long Parameter List, as can be verified by the frequency analysis by systems (Section 4.2). The divergence between the frequencies of occurrence within the methods in different domains suggest that Long Parameter List is also a domain-sensitive bad smell being more common in the health domain.

Health systems usually take into account more variables than systems from other domains since they work with data from patient, health history, diagnostics and even finances within this environment. Therefore, it is expected that methods from the health domain need many parameters to work with.

For the other two bad smells, namely Dead Code and Switch Statements, we do not have an entity associated to them. Therefore, we found suitable to analyze how they are distributed among the domains per KLOC, as we can observe in Figure 7. We can see that five Dead Code instances can be found at every KLOC of accounting systems, being this value the greatest among all domains. Restaurant has almost 3 smells per KLOC while e-commerce and health domains present about 2 Dead Code per KLOC. This information analyzed by itself may indicate that Dead Code is a domain-sensitive smell being more frequent in accounting systems.

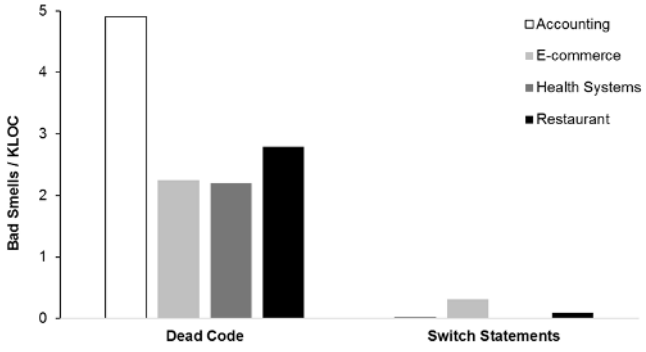

Figure 7: Frequency of Dead Code and Switch Statements in Domains by Entity

As we can observe,there is not a great difference of Dead Code occurrence among all the domains but it is still more frequent in accounting systems. This is expected since bad smells related to code size such as Large Class and Long Method are very common in this domain and then we can have many parts of the code that are no longer used, characterizing the Dead Code.

By looking at Switch Statements, we see values much smaller than for Dead Code. In fact, e-commerce has about 0.31 Switch Statements per KLOC while restaurant and accounting have even smaller number of occurrences (0.09 and 0.01, respectively). Health systems do not present any Switch Statement smell, as already identified in Figure 4. Since the number of occurrence of Switch Statement is very low and it does not have a significant difference among the domains, we cannot infer that this smell is domain-sensitive by looking at the frequency analysis by entity.

4.4 Joint Frequency by System and Frequency by Entity Analysis

This section presents a broader analysis of the frequency from the perspective of systems and entities aiming at summarizing our answers to our research questions (Section 3.1). Table 5 presents an overview of frequency by system and by entity for each bad smell in the four analyzed domains. We define three labels to identify whether a bad smell is possibly sensitive or independent of the software domain. The label âĂIJyâĂİ means that the bad smell is more common in a domain than in others. Similarly, the label âĂIJ(y)âĂİ indicates that the bad smell seems to be more common in a domain (but this relationship is not as strong as that indicated by "y"). We use the label âĂIJnâĂİ to indicate that we could not find any evidence to conclude whether the bad smell is sensitive or independent of the software domain. Columns of Table 5 identifies the six bad smells: Large Class (LC), Long Method (LM), Long Parameter List (LPL), Switch Statements (SS), Dead Code (DC), and Comments (C).

Table 5: Bad Smell Classification per Domain

Frequency by Systems	LC	LM	LPL	SS	DC	C
Accounting	**Y**	**Y**	n	n	n	n
E-commerce	n	(y)	n	**(Y)**	(y)	n
Health	n	**(Y)**	**(Y)**	n	n	n
Restaurant	n	n	n	(y)	n	n

Frequency by Entity	LC	LM	LPL	SS	DC	C
Accounting	**Y**	**Y**	n	n	y	n
E-commerce	n	n	n	**(Y)**	n	n
Health	n	**(Y)**	**(Y)**	n	n	n
Restaurant	n	n	n	n	(y)	n

RQ1 *What are the most frequent bad smells in each software domain?*

In order to answer Research Question 1, we used bold text in Table 5 to identify similar results with respect to the frequency by system and the frequency by entity analyses. For instance, both the frequency by system and by entity analyses suggest that Large Class is more common in accounting systems than in other domains. Therefore, we use bold text in the respective cells. Taking data in Table 5 into account, we summarize our answer to RQ1 as follows.

> **Answering RQ1**. The most frequent bad smells in the account domain are Large Class and Long Method. The most frequent bad smell in the e-commerce domain seems to be Switch Statements. The most frequent bad smell in health systems seems to be Long Method and Long Parameter List. We could not conclude on the most frequent bad smells in the restaurant domain.

RQ2 *What are the domain-independent bad smells?*

Table 5 also supports the identification of domain independent bad smells. Once we have identified the domain-sensitive bad smells, the others are considered domain independent. In other words, the bad smells that are more common in a domain are the domain-sensitive ones. Considering

only the cases of agreement between both frequency by system and by entity, we conclude that Large Class, Long Method, Long Parameter List, and Switch Statements are domain-sensitive bad smells. Therefore, a brief answer to RQ2 can be presented as follows.

Answering RQ2. According to our analysis, the domain-independent bad smells are Dead Code and Comments since they are uniformly distributed within the systems from all analyzed domains.

5. THREATS TO VALIDITY

The focus on this work was to detect and analyze the most common bad smells in specific domains of information systems. In the planning and conduction of this study, some threats may have affected the validity of our research findings. The main issues that threaten the validity of this work are presented and discussed below.

Internal Validity. We identified the following threats to the internal validity: selected domains and key word search strings. We argue that the selected domains are representative, given that they are well-defined in terms of a diversity of recurrent requirements (e.g., user and product management). Therefore, we believe that differences in implementation might reflect in valid varying frequency of bad smells among systems of distinctive domains. Another threat is the reliance on the key word search strings for selecting the initial set of systems in each domain. We cannot ensure that the GitHub search facilities return all relevant systems of each domain. However, we could observe that the search process was able to return systems that we consider relevant to our research questions.

Construction and Conclusion Validity. Threats to the validity also reside on how we have collected and interpreted the results. To avoid problem in data collection, we rely on the default configuration of PMD to automatically detect bad smells. It makes the detection process easy and repeatable. From the perspective of conclusion validity, different interpretations of the results may also represent a threat to the study validity.

External Validity. The major risk here is related to the limitation on the selected systems. First, it is not possible to ensure that they reflect the best samples of the recurrent practice. To reduce this risk, we proceed by selecting systems from GitHub based on the ranking of starred systems. Stars are a meaningful measure for repository popularity, and they may support the selection of relevant and high-quality systems for study. We also excluded systems with less than 1000 lines of code (LOC) because we considered them simple toy examples. Besides, the sample size might be itself another threat to the external validity of the study. We have selected 52 systems from different domains. However, this decision allowed us to obtain more consistent results that could be interpreted in this specific context. Nevertheless, additional replications are necessary to determine if our findings can be generalized to other domains and systems.

6. RELATED WORK

Studies have investigated bad smells in specific domains [4, 8, 19]. For instance, Fontana and colleagues [5] perform an analysis on the impact of bad smells in different domains. Their goal is to identify the most frequent smells in information system domains and to characterize domains with more smells. The authors analyzed 16 bad smells in 68 systems from Qualitas Corpus [21]. They also tried to establish a correlation between bad smells and software quality metrics. Similar to our results, Fontana and colleagues [5] observed that Large Class and Long Method are some of the most common bad smells in general. On the other hand, with respect to bad smells and the domains, they have not observed significant differences among bad smells.

Another related paper, Reis et al. [3] conducted an empirical study with 118 Java systems from 6 information system domains and 7 bad smells. Their goal is to investigate if the domain has a significant impact on the occurrence of bad smells. They observed that most bad smells do not depend on the software domain, with the exception of Duplicated Code. For this bad smell, they showed that its incidence in Home & Education domain was superior to the other domains. Our study differs from Reis research [3] in several ways. First, we rely on PMD tool while Reis used JDeodorant and CodePro AnalytiX. The target systems and domains analyzed in both studies are also different. Therefore, our findings complement the results of Reis and colleagues [3].

Guo et al. [8] also investigated the relations of bad smells and information system domains. However, their focus is on detection rules for bad smells based on software metrics. In other words, they aim to make code smell definitions more accurate and actionable for software developers by tailoring the bad smell definitions to include domain-specific information. Guo and colleagues [8] then enhance a detection tool (CodeVizard) with refinements in the bad smell detections aiming at including domain-specific factors.

In summary, our work follows up previous studies in the investigation of bad smells in information system domains. However our study has several differences form the previous ones. We use different domains in comparison to past works and different bad smells. We also perform an in-deep analysis of the occurrence of bad smells within the domains, since we do an analysis of frequency by entity, such as classes and methods. Finally, We categorize the domain-sensitive and domain-independent bad smells in four information system domains.

7. CONCLUSION AND FUTURE WORK

In this paper, we proposed the identification of domain-sensitive and domain-independent bad smells using PMD tool for analyze analyzing the frequency by systems and by entities in the following information system domains: accounting, e-commerce, health and restaurant. Our findings may bring more awareness for developers of the mentioned software domains and may provide insights to develop more efficient bad smells detection tools taking in account the software domain.

In order to reach our goals, we mined 52 systems from GitHub and analyzed the occurrence of the following bad smells: Large Class, Long Method, Long Parameter List, Switch Statements, Comments, and Dead Code. We used PMD as a bad smell detection tool to find the bad smells. Then, we calculated the total percentage of occurrence considering all systems together and the percentage of each bad smell in relation to the total number of systems within each software domain. Furthermore, we evaluated the bad smell frequency according to the entity with which the smell is related to (class, method or KLOC), what provided us with valuable information about the most and least

frequent bad smells.

This study allowed us to identify domain-independent bad smells whose frequency of occurrence is uniformly distributed among all domain such as COMMENTS and DEAD CODE. We also identified domain-sensitive bad smells that appear more frequently in certain domains when compared to others, like LARGE CLASS, LONG METHOD, LONG PARAMETER LIST, and SWITCH STATEMENTS. Finally, as a suggestion for future work in this context, there is the possibility to expand the amount of systems in each domain as well as to include other domains. This path can be done by automating the analysis of the systems and may bring more chance of generalization of the results and the possibility to do a statistical analysis on the collected data to make the study more reliable.

8. ACKNOWLEDGMENTS

This work was partially supported by CAPES and CNPq (grant 424340/2016-0)

9. REFERENCES

[1] L. Amorim, E. Costa, N. Antunes, B. Fonseca, and M. Ribeiro. Experience report: Evaluating the effectiveness of decision trees for detecting code smells. In *26th Int'l Symposium on Software Reliability Engineering (ISSRE)*, pp.261-269, 2015.

[2] C. Bastos, P. A. Junior, and H. Costa. Detection techniques of dead code: Systematic literature review. In *XII Brazilian Symposium on Information Systems (SBSI)*, pp.255-262, 2016.

[3] J. P. d. Reis, F. B. e Abreu, and G. d. F. Carneiro. Code smells incidence: Does it depend on the application domain? In *10th Int'l Conf. on the Quality of Information and Communications Technology (QUATIC)*, pp.172-177, 2016.

[4] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo. A review-based comparative study of bad smell detection tools. In *20th Int'l Conf. on Evaluation and Assessment in Software Engineering (EASE)*, pp.1-12, 2016.

[5] F. A. Fontana, V. Ferme, A. Marino, B. Walter, and P. Martenka. Investigating the impact of code smells on system's quality: An empirical study on systems of different application domains. In *29th Int'l Conf. on Software Maintenance and Evolution (ICSME)*, pp.260-269, 2013.

[6] F. A. Fontana, E. Mariani, A. Mornioli, R. Sormani, and A. Tonello. An experience report on using code smells detection tools. In *4th Int'l Conf. on Software Testing, Verification and Validation Workshops (ICSTW)*. Computer Society, pp.450-457, 2011.

[7] M. Fowler and K. Beck. *Refactoring: Improving the Design of Existing Code*. Component software series. Addison-Wesley, 1999.

[8] Y. Guo, C. Seaman, N. Zazworka, and F. Shull. Domain-specific tailoring of code smells: an empirical study. In *32nd Int'l Conf. on Software Engineering (ICSE)*, pp.167-170, 2010.

[9] Y. Khrishe and M. Alshayeb. An empirical study on the effect of the order of applying software refactoring. In *7th Int'l Conf. on Computer Science and Information Technology (CSIT)*, pp.1-4, 2016.

[10] R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In *8th Int'l Symposium on Static Analysis (SAS)*, pp.40-56, 2001.

[11] J. Krinke. Identifying similar code with program dependence graphs. In *8th Working Conf. on Reverse Engineering (WCRE)*, pp.301-310, 2001.

[12] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer Publishing Company, Incorporated, 2010.

[13] M. Linares-Vásquez, S. Klock, C. McMillan, A. Sabané, D. Poshyvanyk, and Y.-G. Guéhéneuc. Domain matters: Bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in java mobile apps. In *22th Int'l Conf. on Program Comprehension (ICPC)*. ACM, pp.232-243, 2004.

[14] H. Liu, Z. Ma, W. Shao, and Z. Niu. Schedule of bad smell detection and resolution: A new way to save effort. *Transactions on Software Engineering (TSE)*, (1), pp.220-235, 2012.

[15] W. Ma, L. Chen, Y. Zhou, and B. Xu. Do we have a chance to fix bugs when refactoring code smells? In *Int'l Conf. on Software Analysis, Testing and Evolution (SATE)*, pp.24-29, 2016.

[16] R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *20th Int'l Conf. on Software Maintenance (ICSM)*, pp.350-359, 2004.

[17] N. Moha, Y. g. Gueheneuc, and P. Leduc. Automatic generation of detection algorithms for design defects. In *21st Int'l Conf. on Automated Software Engineering (ASE)*, pp.297-300, 2006.

[18] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, A. D. Lucia, and D. Poshyvanyk. Detecting bad smells in source code using change history information. In *28th Int'l Conf. on Automated Software Engineering (ASE)*, pp.268-278, 2013.

[19] J. Pérez and Y. Crespo. Perspectives on automated correction of bad smells. In *Joint Int'l Workshop on Principles of Software Evolution (IWPSE)*. ACM, pp.99-108, 2009.

[20] N. Sae-Lim, S. Hayashi, and M. Saeki. Context-based code smells prioritization for prefactoring. In *24th Int'l Conf. on Program Comprehension (ICPC)*, pp.1-10, 2016.

[21] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. Qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conf. (APSEC2010)*, pp.336-345, 2010.

[22] N. Tsantalis, T. Chaikalis, and A. Chatzigeorgiou. Jdeodorant: Identification and removal of type-checking bad smells. In *12th European Conf. on Software Maintenance and Reengineering (CSMR)*, pp.329-331, 2008.

[23] C. Wohlin, P. Runeson, M. HÃűst, M. C. Ohlsson, and B. Regnell. *Experimentation in Software Engineering*. Springer, 2012.

[24] S. Wong, Y. Cai, M. Kim, and M. Dalton. Detecting software modularity violations. In *33rd Int'l Conf. on Software Engineering (ICSE)*, pp.411-420, 2011.