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

MARKOS VIGGIATO[1], JOHNATAN OLIVEIRA[2], CLEITON TAVARES[1], EDUARDO FIGUEIREDO[1]

[1]Software Engineering Laboratory (LabSoft)
Department of Computer Science (DCC), Federal University of Minas Gerais (UFMG)
[2]Laboratory of Engineering and Architecture of Systems Distributed in Large Scale (LEASDLE)
Informatics Department, Pontifical Catholic University of Minas Gerais (PUC-MG)
Belo Horizonte (MG) - Brazil
(markosviggiato, figueiredo)@dcc.ufmg.br,
johnatan.oliveira@sga.pucminas.br, cleitonsilvatavares@gmail.com

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 six specific information systems domains: accounting, e-commerce, health, games, dictionaries and restaurant. Our analysis relies on 88 information systems mined from GitHub. We identified bad smells with two detection tools, PMD and Checkstyle. Our findings suggest that COMMENTS is a domain-independent bad smell since it uniformly appears in all investigated domains. On the other hand, LARGE CLASS and LONG PARAMETER LIST can be considered domain-sensitive bad smells since they appear more frequently in accounting and health systems, respectively. Although less frequent in general, SWITCH STATEMENTS also appear more in health systems than in other domains.

Keywords: Bad Smell, Information System Domain, Detection Tools.

(Received September 16th, 2017 / Accepted October 18th, 2017)

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 88 object-oriented Java information systems mined from GitHub of 6 different domains: accounting, e-commerce, health, restaurant, games and dictionaries. We rely on PMD [5] and Checkstyle[1] to detect 6 types of bad smells in the target information systems. 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 can be considered domain-sensitive bad smells since it appears more frequently in accounting systems. In this study, we also observe that LONG PARAMETER LIST and SWITCH STATEMENTS have lower values of frequency when compared to the other bad smells. Although less frequent, LONG PARAMETER LIST and SWITCH STATEMENTS are more common in health systems than in other ones.

This paper is an extension of a previous work [22], where we performed a preliminary study in order to identify the frequency of occurrence of different bad smells in systems from different information system domains.

The current work brings many additional contributions. Here, we analyze a larger number of systems, previously we had 52 and now we have 88 systems. In addition, we also included two new domains, games and dictionaries. In this way, we perform our analysis over 88 Java systems mined from GitHub. Also, we now use two completely automated detection tools (PMD and Checkstyle), instead of one, to identify bad smells within the systems, what brings more reliability to our results, since we have more than one source of information regarding the detection of bad smells.

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 one of the tools we used: PMD and Checkstyle. In total, our analysis relies on 6 types of bad smells: LARGE CLASS, LONG METHOD, LONG PARAMETER LIST, SWITCH STATEMENTS, COMMENTS, and DEAD CODE.

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

!h

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 "if...else", statement) is scattered in a program in many different places.

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 Checkstyle. Their characteristics can be verified in Table 2 and are detailed as follows.

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.

Checkstyle is an open-source tool that automatically identifies 4 bad smells [4]: LARGE CLASS, LONG METHOD, LONG PARAMETER LIST and DUPLICATED CODE. Similarly to PMD, Checkstyle uses software metrics and thresholds to detect bad smells. PMD and Checkstyle 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]. 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, games, dictionaries 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.

Table 2: Bad smells Detection Tools

Features	Checkstyle	PMD
Type	Eclipse plug-in and Standalone	Eclipse plug-in and Standalone
Version	7.5 / 2016	5.5.4 / 2017
Supported Languages	Java	C,C#, C++, JAVA PHP and 11 others
Bad Smells Detected	Large Class, Long Method, Duplicated Code, and Long Parameter List	Dead Code, Comments, Large Class, Long Method, Duplicated Code, and Long Parameter List

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.

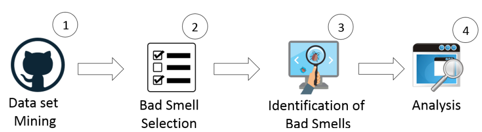

Figure 1: Study Phases

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, restaurant, game and dictionary. 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 Checkstyle.

Phase 3) Identification of bad smells - In order to make the analysis of a large number of system feasible, we automated the two detection tools. Therefore, we only need to define the detection rules in a .xml file for each tool, so that we can determine specifically which bad smells the tools are able to identify. We then execute both tools passing the .xml file containing the rules and the directory where the systems are located. The tools' output consists of a .csv file with the identified bad smells.

Phase 4) Analysis - We cloned the information systems of the six domains from GitHub. We then 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.

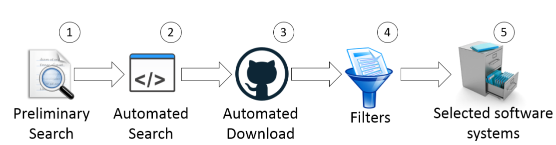

Figure 2: Steps for Collecting Systems from GitHub

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
Games	Game OR Games
Dictionaries	Dictionary OR Dictionaries
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 600 information systems, 100 for each domain sorted in descending order by stars. Here, we complement the previous study [22], where we obtained 400 systems. Therefore, in this work we extended our dataset to 600 systems, downloading 200 additional systems. 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 480 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 230 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 88 systems). After applying the filters presented in step (4), we obtained 88 systems. Therefore, in step (5), our data set is finally composed by 88 Java information systems to support the identification of the bad smells. The number of systems from each domain varies from 10 to 20.

To better characterize our data set, we computed the metrics from Metrics[2] 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 dictionaries domain is the biggest one in code size, having more than 365 KLOC, while restaurant is the smallest one, with less than 51 KLOC.

Table 4: Data Set Characterization

Software Domains	# Systems	LOC	NOC	NOM	NOA
Accounting	12	74,180	472	5,832	4,567
E-commerce	19	57,366	832	5,531	2,277
Health	14	237.738	1.506	14,596	6,227
Games	14	235,254	3.496	24,605	9.815
Dictionaries	14	365,455	3,685	38,333	15,306
Restaurant	15	50,105	531	4,525	2,307

4 Results

This section presents the results and analysis. We analyzed 88 Java systems mined from GitHub from accounting, e-commerce, health, games, dictionaries and restaurant domains. Six bad smells were analyzed in these systems with the PMD and Checkstyle detection tools. 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, being identified in more than 50% of the systems. In fact, we detected COMMENTS in all systems of our data set, so this is the most common bad smell in all the systems.

According to our data, the second most common bad smell is DEAD CODE. This bad smell was detected in approximately 78% of the systems in all domains. LARGE CLASS and LONG METHOD are also very common since they could be found in about 67% and

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

64% of the systems, respectively. On the other hand, LONG PARAMETER LIST and SWITCH STATEMENTS are the least frequent bad smells, appearing, respectively, in about 29% and almost 41% of the systems.

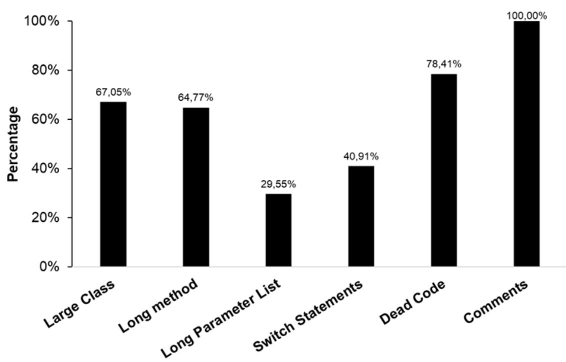

Figure 3: Percentage of Systems with Bad Smells

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. It is important to note that we excluded COMMENTS from this analysis because we observed that it is highly frequent in all domains (Section 4.1). In fact, COMMENTS was found in all systems of every domain, therefore its frequency is 100% for all domains. 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, games and health systems. In fact, LARGE CLASS is present in 92% of accounting systems, in 86% of games systems and in about 71% of health systems. Similarly, LONG METHOD was identified in 92% of accounting systems and 93% and 71% of games and health systems, respectively. 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 47% of e-commerce and restaurant domains and about 70% in dictionaries, while LONG METHOD was found, respectively, in 47%, 40% and about 55% for these domains. Therefore, it is clear that both LARGE CLASS and LONG METHOD seem to be domain-sensitive bad smells with the highest frequencies in accounting, games and health domains, and low frequencies in the other domains.

Figure 4 also shows interesting results for LONG PARAMETER LIST. PMD and Checkstyle found this bad smell in about 58% of systems in the health domain. On the other hand, only 29% of games and dictionaries systems have LONG PARAMETER LIST, while the other domains have frequency even lower. This large difference suggests that LONG PARAMETER LIST is a domain-sensitive bad smell with the highest frequency in health systems. Our data so far do not allow us to say whether DEAD CODE and SWITCH STATEMENTS are domain-independent or domain-sensitive bad smells since they do not present such large difference in frequency of occurrence among all domains.

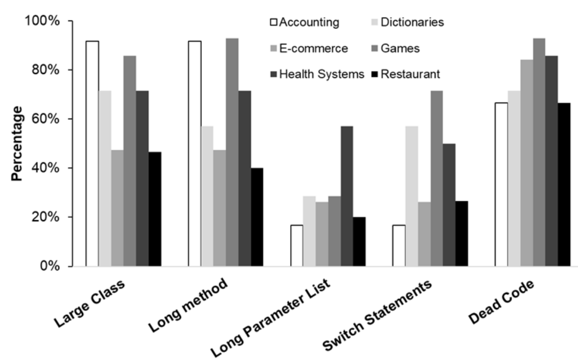

Figure 4: Frequency of Bad Smells in Software Domains

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. Since the analyzed bad smells have different granularities (e.g., class-level and method-level), we have different entities. 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 according to their distribution within the code using KLOC.

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.

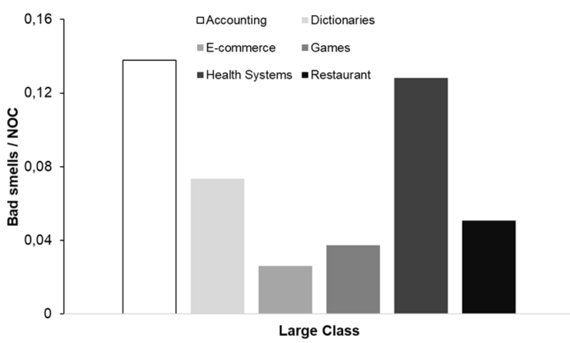

Figure 5: Frequency of LARGE CLASS in Domains by Entity

Data in Figure 5 show that LARGE CLASS has the highest frequency of occurrence in the accounting domain, followed by the health one, in comparison to other domains. In fact, it is present in approximately 0.14 (14%) of the classes of accounting and 0.13 (13%) of the classes of health systems, i.e., 14% of the classes of accounting and 13% of the classes in health domain are LARGE CLASSES.Dictionaries and restaurant have about 7% and 5%, respectively, of LARGE CLASSES, while games and e-commerce present the lowest values of frequency (about 3% and 2%, respectively). The significant difference between accounting and health and the other domains suggests that LARGE CLASS is a domain-sensitive bad smell appearing more frequently in the accounting and health domains.

We believe that the high frequency of LARGE CLASS in those two domains is due the number of calculations done in accounting systems and the diverse features present in specific classes of health systems, 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 in the accounting domain, for example, 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 health domain being present in about 2.1% of all methods. Next, we have accounting and restaurant domains, with frequencies of 1.6% and 1.2%, respectively. For the other domains, we can observe very low frequencies of occurrence, all below than 0.8%. These numbers may indicate that LONG METHOD is a domain-sensitive bad smell since it appears more frequently in health, accounting and restaurant systems than in others, what is in line with the frequency analysis by systems presented in Section 4.2, since LONG METHODS are highly frequent in some domains and have low rates of occurrences in other domains.

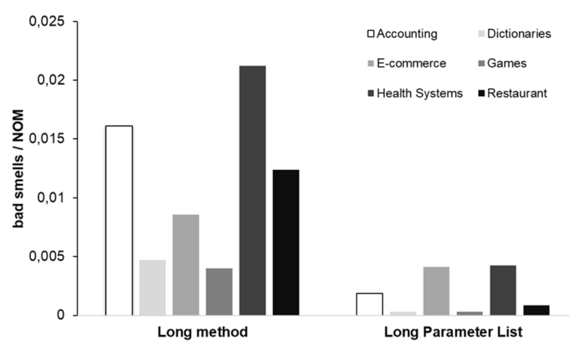

Figure 6: Frequency of LONG METHOD and LONG PARAMETER LIST in Domains by Entity

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. The same is valid for health systems, since the various features of the systems are located within the methods, making them grow in code size and also in complexity.

LONG PARAMETER LIST occurs more frequently in health and e-commerce systems with a percentage of 0.4% for both of them, followed by accounting with only 0.1%. We can note that the other domains have very low values of frequency, being the largest one the restaurant with 0.08% of LONG METHODS. 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 and e-commerce domains.

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. E-commerce also needs a large number of parameter per method, such as information about products, sales and stock status.

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 about 9 Dead Code instances (e.g., unused variable or method) can be found at every KLOC of accounting systems, being this value the greatest among all domains. Restaurant has about 3 smells per KLOC while the other domains present lower number of Dead Code per KLOC. This information analyzed by itself may indicate that Dead Code is a domain-sensitive smell being much more frequent in accounting systems.

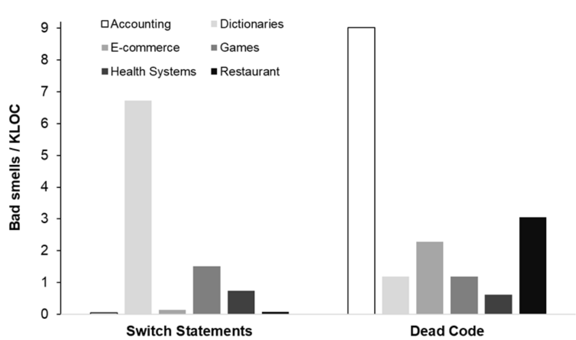

Figure 7: Frequency of Dead Code and Switch Statements in Domains by Entity

As we can observe, apart from accounting domain, there is not a great difference of Dead Code occurrence among the other domains. This is expected since bad smells related to code size such as Large Class and Long Method are very common in accounting systems 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 smaller values than for Dead Code. In fact, dictionaries domain has the highest value of frequency (almost 7 Switch Statements per KLOC), while the other domains have much lower values, being almost zero for accounting systems. Since the number of occurrence of Switch Statement is much higher for dictionaries than for the other domains, this information by itself may indicate that Switch Statement is a domain-sensitive bad smell. However, we can note that, apart from dictionaries, the frequencies for all the other domains are very similar, therefore we cannot surely infer that this smell is domain-sensitive since accounting domain may represent a singular situation.

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 six analyzed domains. We define three labels to identify whether a bad smell is possibly sensitive or independent of the software domain. The label “y” means that the bad smell is more common in a domain than in others. Similarly, the label “(y)” 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 “n” 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	n	**(Y)**	n
Health	**(Y)**	(Y)	**(Y)**	**(Y)**	(Y)	n
Games	Y	Y	n	Y	(Y)	n
Dictionaries	(Y)	n	n	(Y)	n	n
Restaurant	n	n	n	n	n	n

Frequency by Entity	LC	LM	LPL	SS	DC	C
Accounting	**Y**	(Y)	n	n	y	n
E-commerce	n	n	(Y)	n	**(Y)**	n
Health	**(Y)**	Y	**(Y)**	**(Y)**	n	n
Games	n	n	n	(Y)	n	n
Dictionaries	n	n	n	Y	n	n
Restaurant	n	(Y)	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 and health 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 smell in the account domain is LARGE CLASS. The most frequent bad smell in the e-commerce domain seems to be DEAD CODE. The most frequent bad smells in health systems seems to be LARGE CLASS, LONG PARAMETER LIST and SWITCH STATEMENTS. We could not conclude on the most frequent bad smells in the restaurant, games and dictionaries domains.

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 PARAMETER LIST, SWITCH STATEMENTS and DEAD CODE 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 LONG METHOD 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 PMD and Checkstyle to automatically detect bad smells. 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 88 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 and Checkstyle tools 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 six 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 and Checkstyle detection tools to analyze the frequency by systems and by entities in the following information system domains: accounting, e-commerce, health, games, dictionaries 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 into account the software domain information.

In order to reach our goals, we mined 88 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 and Checkstyle as bad smell detection tools 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 LONG METHOD. We also identified domain-sensitive bad smells that appear more frequently in certain domains when compared to others, like LARGE CLASS, LONG PARAMETER LIST, SWITCH STATEMENTS and DEAD CODE. Finally, as a suggestion for future work in this context, we intend to analyze the agreement between the detection tools, providing their precision e recall. There is also the possibility to expand the amount of systems in each domain as well as to include other domains. This path can be done since the analysis of bad smells are already automated 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)

References

[1] Amorim, L., Costa, E., Antunes, N., Fonseca, B., and Ribeiro, M. 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] Bastos, C., Junior, P. A., and Costa, H. Detection techniques of dead code: Systematic literature review. In *XII Brazilian Symposium on Information Systems (SBSI)*, pp.255-262, 2016.

[3] d. Reis, J. P., e Abreu, F. B., and d. F. Carneiro, G. 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] Fernandes, E., Oliveira, J., Vale, G., Paiva, T., and Figueiredo, E. 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] Fontana, F. A., Ferme, V., Marino, A., Walter, B., and Martenka, P. 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] Fontana, F. A., Mariani, E., Mornioli, A., Sormani, R., and Tonello, A. 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] Fowler, M. and Beck, K. *Refactoring: Improving the Design of Existing Code*. Component software series. Addison-Wesley, 1999.

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

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

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

[12] Lanza, M. and Marinescu, R. *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] Linares-Vásquez, M., Klock, S., McMillan, C., Sabané, A., Poshyvanyk, D., and Guéhéneuc, Y.-G. 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] Liu, H., Ma, Z., Shao, W., and Niu, Z. Schedule of bad smell detection and resolution: A new way to save effort. *Transactions on Software Engineering (TSE)*, (1), pp.220-235, 2012.

[15] Ma, W., Chen, L., Zhou, Y., and Xu, B. 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] Marinescu, R. Detection strategies: Metrics-based rules for detecting design flaws. In *20th Int'l Conf. on Software Maintenance (ICSM)*, pp.350-359, 2004.

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

[18] Palomba, F., Bavota, G., Penta, M. D., Oliveto, R., Lucia, A. D., and Poshyvanyk, D. 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] Pérez, J. and Crespo, Y. Perspectives on automated correction of bad smells. In *Joint Int'l Workshop on Principles of Software Evolution (IWPSE)*. ACM, pp.99-108, 2009.

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

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

[22] Viggiato, M., Oliveira, J., and Figueiredo, E. On the investigation of domain-sensitive bad smells in information systems. In *13th Brazilian Symposium of Information Systems (SBSI)*, pp.388-395, 2017.

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