# On the Interplay of Smells *Large Class*, *Complex Class* and *Duplicate Code*

Elder Vicente de Paulo Sobrinho
Federal University of Triângulo Mineiro
Uberaba, MG, Brazil
elder.sobrinho@uftm.edu.br

Marcelo de Almeida Maia
Federal University of Uberlândia
Uberlândia, MG, Brazil
marcelo.maia@ufu.br

## ABSTRACT

Bad smells have been defined to describe potential problems in code, possibly pointing out refactoring opportunities. Several empirical studies have highlighted that smells have a negative impact on comprehension and maintainability. Consequently, several approaches have been proposed to detect and restructure them. However, studies on the inter-relationship of occurrence of different types of smells in source code are still lacking, especially those focused on the quantification of this inter-relationship. In this work, we aim at understand and quantify the possible the inter-relation of smells *Large Class - LC, Complex Class - CC* and *Duplicate Code - DC*. In particular, we investigate patterns of LC and CC regarding the presence or absence of duplicate code. We conduct a quantitative study on five open source projects, and also a qualitative analysis to measure and understand the association of specific smells. As one of the main results, we highlight that there are "occurrence patterns" among these smells, for example: either in *Complex Class* or in the co-occurrence of *Large Class* and *Complex Class*, clones tend to be more prevalent in highly complex classes than less complex classes. The found patterns could be used to improve the performance of detection tools or even help in refactoring tasks.

## KEYWORDS

Software maintenance, reengineering, bad smell

## 1 INTRODUCTION

Software systems need to evolve continuously to cope with new requirements and environment changes. High-quality source code plays an important role in this context because the code itself is required to be easy to understand, analyze, change, maintain, and reuse [11]. However, software developers eventually produce sub-optimal code (not at the highest standard), possibly introducing design problems, i.e., they produce code structures that violate fundamental principles in software engineering, such as, high cohesion and low coupling. Bad smells have been proposed as a metaphor for sub-optimal code structures, and have gained attention after Fowler and Beck [8] proposing that those structures could be refactored in a systematic way to produce better quality code. On the other hand, bad smells may not be as harmful as generally claimed, in other words, they are not always associated with undesirable or problematic situations [21]. For instance, Rahman et al. [17] report that bugs are not significantly associated with *Duplicated Code*. Also, in some situations, writing code with the presence of bad smells is even the best option for developers [26].

Despite the large body of knowledge already produced on code bad smells, there is still room for investigating better this topic [21]. There may be common sense knowledge that may not hold as expected in real-world projects. An interesting example has been shown by Tufano et al. [24], where their findings *"contradict common wisdom, showing that most of the smell instances are introduced when an artifact is created and not as a result of its evolution"*. In our work, we start from the observation that smells seems to be a fragmented metaphor for analyzing code entities because they are defined to characterize a single sub-optimal structure in the code. However, code entities do not manifest only a pure view of such metaphors, i.e., an entity may manifest more than one kind of smell simultaneously. So, in this work, we target a phenomenon that is not much addressed in the literature: the co-occurrence of different smell types in the same code entity. We aim at investigating the inter-relation between multiple kinds of bad smells. In particular, we study the inter-relation of smells *Duplicate Code* (DC), *Large Class* (LC) and *Complex Class* (CC). For instance, we aim at verifying the actual extent of common wisdom raised by Fowler and Beck [8], when they explain the smell *Large Class* and suggest an interplay with *Duplicate Code*: *"...a class with too much code is prime breeding ground for duplicated code..."*. Moreover, there could be other scenarios, for instance, where complex and large classes could contain code snippets cloned in other also complex and large classes. This potential pattern could be possibly useful for software engineers, in particular, to improve their rules of code review (e.g., check the possible existence of clones in complex and large classes to improve three kinds of sub-optimal code structures simultaneously). Summing up, we expect that such possible inter-relationships could be unveiled by analyzing how the co-occurrence of complex and large class smells may actually affect the prevalence of clones.

The smells *Duplicate Code*, *Large Class* and *Complex Class* are the focus of this paper because: i) these smells are recurrent in literature but, to the best of our knowledge, no paper has investigated their inter-relation; ii) those smells are commonly found in the source code of several projects and the number of instances of them is large enough to allow statistical analysis; iii) semantically there seems to be a relationship between themselves, so we aim to investigate if the following hypothesis actually holds: the complex entities with many control flow statements could be prone to the occurrence of clones, in particular, when a large number of conditional expressions are present and perform the same code or slightly different codes (differing only in their conditions)[1]. Moreover, there is still the fact that the code smell metaphor has been proposed to deliver a boolean value, e.g., a class is large or not. However, the intensity of a smell may indicate its severity. For instance, Palomba et al. [16]

[1]https://refactoring.guru/smells/duplicate-code

arXiv:2107.09512v1 [cs.SE] 20 Jul 2021

have improved a bug prediction model adding the intensity of the smell as a predictor. In this paper, we study the interplay of smells according to their co-occurrence and intensity in different combinations. The main contributions of the paper are: i) we expand the knowledge about the prevalence of the co-occurrence of *Duplicate Code*, *Large Class* and *Complex Class*. In particular, we investigate patterns of *LC* and *CC* regarding the presence or absence of duplicate code; ii) we reveal the occurrence of patterns among smells, e.g., the prevalence of clones is much associated to the occurrence of the smell *Complex Class* than to the isolated occurrences of *Large Class*; iii) to the best of our knowledge, this is the first work to use intensity of smells to finding patterns among several smells; iv) we also introduce practical implications of our results on strategies used to detect smells and on to refactoring planning.

**Structure.** The remainder of this paper is structured as follows: in the next section, we present background information and review the related work. The Section 3 describes the design of our empirical study, while Section 4 reports the obtained results. Next, the Section 5 discusses and provides a qualitative perspective of our results. In Section 6, we present the limitations and threats. Finally, in Section 7, our conclusions are drawn.

## 2 BACKGROUND AND RELATED WORK

Next, we present some concepts used and discuss related work.

### 2.1 Background

We use the concept of interrelationship and intensity of smells. First, we show different types of interrelationships, and then we define a metric to measure how intense each smell instance is.

#### 2.1.1 Interrelationship of smells.

Smells could be interrelated in several different manners. We present three possible different forms of smell interrelationship: *by co-occurrence, by static dependency* and *by commit dependency*. In our work, we consider only the interrelationship by co-occurrence.

The class *C1* (Figure 1) has three different kinds of smells ($S1_C$, $S2_C$, $S3_C$) and the class *C2* show another kind of smell ($S4_C$). These two entities have four smells and they occur at the level of class. In the method level, these classes have other three kinds of smells ($S5_M$, $S6_M$, $S7_M$). Observe that the method *M1* and *M2* has the same kind of smell ($S6_M$).

Smells can be interrelated by the source code structure, either by a co-occurrence or by a static dependency. The interrelationship by *co-occurrence* considers the existence of smells only in one entity at a time, e.g., the file of the class *C1* have five smells ($S1_C$, $S2_C$, $S3_C$, $S5_M$, $S6_M$) and they can be grouped by the granularity of entity (class/method). The interrelationship by *static dependency* takes into account the coupling with other entities and the existence of smells in them, e.g., the method *M1* inside the class *C1* invokes the methods *M2* and *M3* of the class *C2*. Thus, these seven smells ($S1_C$, $S2_C$, $S3_C$, $S4_C$, $S5_M$, $S6_M$, $S7_M$) are interrelated by a *static dependency* of the respective entities, and they can be grouped by granularity of each entity. The change history of source code could also be used to interrelate smells, in particular, using *commits*. Considering the methods *M2* and *M3*, there is no interrelationship by the source code structure. Nevertheless, taking into account *commits*, if those methods were possibly changed inside the same

commit, they would be commit-interrelated, meaning that possibly smells $S6_M$, $S7_M$ would have some impact on *M3*.

Considering that the smells can be interrelated in different forms, we clarify that in this paper only the interrelationship of smells by *co-occurrence* at the class level are studied.



**Figure 1: Interrelationship of smells (co-occurrence & dependency).**

#### 2.1.2 Intensity of Smells.

Steidl and Eder [23] describes an approach to compute smell intensity based on refactoring techniques (*pull-up method, extract method*) used to remove the smells. They compute the smell intensity, considering the size (LOC) and the number of parameters necessary to complete the refactoring task. Fontana et al. [7] consider the threshold of metrics used on smell detection to classify the level of intensity of smells. On their approach, the more the value of a given metric exceeds your threshold value, the greater is the smell intensity. Based on the distribution of these metrics, they classified the smell intensity in five range (*Very-Low, Low, Mean, High, Very-High*).

We also consider the smell intensity on this study of interrelationship of smells. The intensity of smells could possibly improve the identification of patterns, e.g., as the smells become more critical, they could possibly end up associating themselves with other types of smells. Thus, in context of smell *Complex Class*, we can analyze whether when complexity increases, the prevalence of other types of smells also increases.

We use an approach similar that proposed by Fontana et al. [7] to compute the intensity of smells, but we simplify on only two ranges (*Low* or *High*). We decided not to use a continuous variable for intensity because it would be more difficult to find any pattern using such representation, so a binary representation would have more chances to explain possible patterns. Let $E = \{e_{1_{S_1}}, e_{2_{S_2}}, ..., e_{n_{S_n}}\}$ be a set of distinct entities $e_{i_{S_i}}$, where in each one co-occurs a set $S_i$ of different smells ($1 \leq i \leq n$). Let $S = \bigcup_{i=1}^{n} S_i$ be the set of all distinct smells. Let $s_j$ be a smell in $S$. In order to compute the intensity of a smell $s_j$ in an entity $e_i$, we extract a subset $E_{s_j}$ of $E$ with all entities of $E$ occurring $s_j$, i.e., $E_{s_j} = \{e_{S'} \mid e_{S'} \in E \wedge s_j \in S'\}$. Then, let the set $M_j$ contain the computed metrics[2] $m_j(e)$ used to identify this smell $s_j$ in each $e \in E_{s_j}$. Finally, for each entity $e_i$ ($1 \leq i \leq n$) with smell $s_j$ ($1 \leq j \leq S$), we use the Equation 1 to classify their intensities.

$$Intensity_{s_j}(e_i) = \begin{cases} High, & if \ m_j(e_i) \geq median(M_j) \\ Low, & otherwise \end{cases} \quad (1)$$

---

[2] *Complex Class* = *Cyclomatic Complexity*, *Large Class* = Number of methods declared + Number of attributes declared, both used by DECOR. *Duplicate Code* = number of tokens, used by PMD tool.

## 2.2 Related Work

According to Oizumi et al. [13] each smell alone may represent only a partial embodiment of a design problem. They suggest that smells tend to "flock together" to realize a design problem. Thus, they investigated whether and how smelly code relationships can help developers to locate design problems. To achieve this purpose, they propose a strategy to identify groups of inter-related smells. This strategy is composed of syntactic and semantic forms used to connect elements of a software, as example: two smells are syntactically related if their host program elements are connected through method calls or inheritance relationships. Their analysis indicates that certain forms to connect elements are consistent indicators of both congenital and evolutionary design problems, with accuracy often higher than 80%. They also found the combined use of syntactic and semantic forms to connect elements of a software represents a more effective approach for locating design problems.

Recent studies suggest that developers should ignore smells occurring in isolation. Instead, they should focus on analyzing smell agglomerations, e.g., a entity affected by multiple smells. However, there is limited understanding if developers can effectively identify a design problem in stinkier code. Developers may struggle to make a meaning out of inter-related smells affecting the same program location. In this context, Oizumi et al. [14] applied an approach to analyze if and how developers can effectively find design problems when a program location is affected by multiple smells. The analysis revealed that only 36.36% of the developers found more design problems when they explicitly use smell agglomerations to identify design problems as compared to single occurrence of smells. On the other hand, 63.63% of the developers reported much lesser false positives. Developers reported that analyses of smell agglomerations scattered in class hierarchies or packages are often difficult, time consuming, and requires proper visualization support. Moreover, it remains time-consuming to discard stinky program locations that do not represent design problems.

Santana et al. [19] have also characterized co-occurrences as agglomerations, i.e., when two or more bad smells occur in the same snippet of code. They studied four kinds of bad smells: Large Class, Long Method, Feature Envy and Refused Bequest using association rules and found that classes with two or more smells are frequent in the source code, even when the smells in a class are of the same type. They also found that agglomerations are highly spread in the source code having significant effect on modularity metrics.

Fontana et al. [6] focus their attention on the possible relations existing among code smells and their co-occurrence (how many entities are affected by more than one smell), with the aim to find and detect some architecturally relevant code smells. They found that a significant percentage of the detected instances has a relation with other instances. For example, 26% of *God Classes* use at least a *Data Class*, and that 53% and 70% of methods affected by *Shotgun Surgery* and *Dispersed Coupling*, respectively, are called from (at least) one class or method affected by a code smell. This observation confirms other results and theories proposed in different studies, suggesting that code smells tend to cluster together and interact in many ways, and that clusters of smells have more effect on maintainability than isolated smells.

According to Palomba et al. [15], there is little knowledge regarding which smell types tend to co-occur in code. To enlarge the knowledge on the phenomenon, they provide a large-scale replication of previous studies, taking into account 13 smell types on a dataset composed of 395 releases of 30 software systems. They identified six pairs of code smells that co-occur very often, some of these co-occurrences are quite expected (e.g., *Long Method* and *Spaghetti Code*), others are not (e.g., *Message Chains* and *Refused Bequest*), recalling the need for studying more deeply the reasons behind their appearance and their apparent relationships.

Jaafar et al. [10] investigated the relationship between code clones and 15 anti-patterns (smells) documented by Brown et al. [2]. They conducted the study on three open-source systems and results show that clones and anti-patterns is a frequent observation: at least, more than 52% of anti-pattern classes have clones, while 59% to 78% of classes with clones are participating in anti-patterns. They also observe that classes having clones and anti-patterns are significantly more fault-prone than other classes. The analysis also reveals that a high number of smell co-occurrence among classes increases the risk of fault induction in software systems and decrease their reliability. Finally, they suggest that the detection of smell co-occurrence among classes helps to manage change commits and to avoid faults induction during the maintenance activities.

Yamashita and Moonen [27] investigated interactions (e.g., smells that were co-located in the same artifact) among twelve different smells and how these interactions can lead to maintenance problems. Analyzing how developers conducted tasks on four different systems, they found evidence that certain inter-smell relations were associated with problems during maintenance, e.g.: i) artifacts containing *ISP Violation* relates to the presence of *inconsistent design*; ii) classes contained many methods that accessed data/methods from different areas of the system (i.e., methods displaying *Feature Envy*) are related to the faults because developers missed areas of the code that needed to be consistently changed after changes were done on the methods displaying *Feature Envy*.

Previous studies have not investigated the relation between the smells *Duplicate Code*, *Large Class* and *Complex Class*. Moreover, studies that take into account the intensity of these smells are also lacking. Thus, in this study we sought to provide additional evidence on the relationship between some kinds of smells.

## 3 STUDY SETTING

In this section we define the research questions of the study, explaining the process followed to assess them.

## 3.1 Research Questions

The *goal* of the study is to analyze the interplay of co-occurrence of smells *Duplicate Code (DC)*, *Large Class (LC)* and *Complex Class (CC)*. The motivation for this analysis is to assess to which extent the several combinations of smells *LC* and *CC* in different combinations of (co-)occurrence associates with cloning in the respective entities. To organize the analysis, the study has been divided into two parts: i) analysis considering the absolute frequency of smells; ii) analysis considering the intensity of smells.

In the first part of our research questions, we investigate what happens with the prevalence of clones as the frequency of *CC* and *LC* smells co-occurring in a class changes.

**RQ1.1** Is there a difference in the prevalence of clones comparing the entities where only *CC* occur and those entities where *CC* and *LC* co-occur? If so, how much this difference is associated to the smells *CC* and/or *LC*?

As shown in Figure 2, we compare two groups of classes: 1) on the left, all classes with smells *CC*, not occurring the smell *LC* and 2) on the right, all classes with co-occurrence of smells *CC & LC*. In both groups, we have classes with and without the smell *DC*.

**RQ1.2** Is there a difference in the prevalence of clones comparing the entities where *LC* and *CC* co-occur and those entities where only *LC* occur? If so, how much this behavior is associated to the smells *LC* and/or *CC*?

Similarly to the previous RQ as also shown in Figure 2, we also compare two groups of classes: 1) on the left, all classes with smells *LC* and that does not have the smell *CC* and 2) on the right, all classes with co-occurrence of smells *CC & LC*. Again, both groups have classes with and without the smell *DC*.
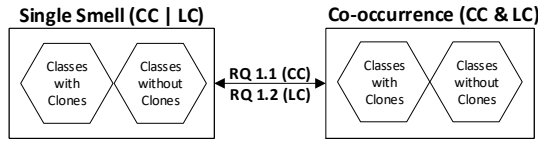


**Figure 2: Diagram of RQ1.1 and RQ1.2.**

In the second part of our research questions, we evaluate what happens to the clone prevalence when the intensity of a smell or a set of smells becomes more critical.

**RQ2.1** Is there any association between the prevalence of clones and the intensity of smell *Complex Class*?

This research question investigates the relationship between the occurrence of clones and entities classified as less complex and those more complex. Figure 3 shows that, for RQ2.1, we compare two groups of classes, 1) on the left, all classes with only smell CC in *Low intensity* and 2) on the right, all classes with this smell in *High intensity*. In both groups, we have classes with and without smell DC. Additionally, in these RQs, we do not consider the co-occurrence of smells (CC & LC).

**RQ2.2** Is there any association between the prevalence of clone and the intensity of smell *Large Class*?

Similarly to the previous RQ2.1, but now for *LC*, this question investigates the relationship between the occurrence of clones and entities classified as *Large Class* only. This relationship is compared under two levels of *Large Class* intensity (*Low* and *High*).

**RQ2.3** Is there any association between the prevalence of clone and the intensity and co-occurrence of smells *Large Class* and *Complex Class*?
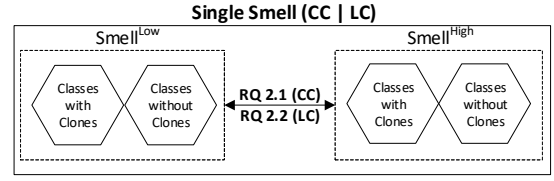


**Figure 3: Diagram of RQ2.1 and RQ2.2.**

This research question investigates the relationship between the occurrence of clones and co-occurrence of smells *Large Class* and *Complex Class*, which are analyzed in two levels of intensity (*Low* vs. *High*). As shown in Figure 4, we formulated two models based on smell co-occurrence and intensity to analyze the impact of the intensity of *Large Class* and *Complex Class*: $CC^{Low}$ & $LC$ vs $CC^{High}$ & $LC$ and $CC$ & $LC^{Low}$ vs $CC$ & $LC^{High}$.
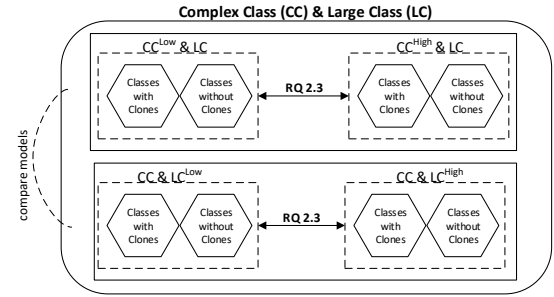


**Figure 4: Diagram of RQ2.3.**

## 3.2 Study Variables

The **dependent variables** considered in our study, for all the research questions, are the presence or absence of clones being observed across different software. In other words, this variable is dichotomous ($Y_{DC}$) and denotes the presence of smell *Duplicate Code*. The **independent variables** are the factors related to the smell intensity (LC,CC), single occurrence of smell (LC,CC) and/or co-occurrence of smells (LC,CC) and are defined according to the research questions:

**RQ1.1**. We have a categorical variable with two possible values: one denoting the occurrence of the smell *Complex Class* only and another denoting the co-occurrence of smells *Complex Class* and *Large Class*. The model is: $Y_{DC} = \beta_0 + \beta_1 X_{Single\_Co\text{-}occurrence}$

**RQ1.2**. Similar to the previous question, we also have a categorical variable limited to two values: one denoting the occurrence of the smell *Large Class* only and another denoting the co-occurrence of smells *Complex Class* and *Large Class.*. Thus, the model also is similar.

**RQ2.1**. For this research question, the categorical variable denotes the intensity of the smell *Complex Class* in two levels (*Low|High*). The model may be represented as: $Y_{DC} = \beta_0 + \beta_1 X_{Low\_High}$.

**RQ2.2**. In this case, the categorical variable $X$ denotes the intensity of the smell *Large Class*. Thus, the model is similar to the previous question.

**Table 1: Studied open source projects.**

| Project | Version | Java File | LOC | Snippets Cloned | Classes With Clones | LARGE CLASS | COMPLEX CLASS | Co-occurrence LC & CC |
|---------|---------|-----------|-----|-----------------|---------------------|-------------|---------------|-----------------------|
| ArgoUML | 0.34 | 1233 | 105795 | 214 | 79 | 63 | 37 | 119 |
| Cassandra | 3.11 | 2062 | 333211 | 760 | 31 | 38 | 44 | 110 |
| Lucene | 6.2.1 | 3848 | 533926 | 2255 | 137 | 54 | 100 | 160 |
| Hadoop | 2.6.0 | 1417 | 194518 | 605 | 25 | 26 | 25 | 68 |
| Ant | 1.8.2 | 1182 | 127042 | 170 | 33 | 10 | 33 | 96 |

**RQ2.3**. The categorical variable denotes the intensity (*Low*|*High*) of two smells (*Complex Class* and *Large Class*). In particular, the intensity of these smells are investigated when they co-occur in the same class. Thus, we have two models: i) considering the intensity of the smell *Large Class* ($Y'_{DC} = \beta_0 + \beta_1 X_{CC\&LC_{Low\_High}}$), and ii) based on the intensity of the smell *Complex Class* ($Y''_{DC} = \beta_0 + \beta_1 X_{LC\&CC_{Low\_High}}$).

## 3.3 Studied Systems and Data Extraction

The study consists of 5 Java open source software systems and having different sizes and domains (see Table 1). ArgoUML is a UML diagramming system in Java. Lucene is a a high-performance, full-featured text search engine library. Cassandra is a database management system. Hadoop is a tool for distributed computing. Apache Ant is a build tool and library specifically conceived for Java applications (though it can be used for other purposes).

To answer our research questions, we first need to detect the smells of the studied systems. To this aim we use existing tools, PMD[3] and DECOR[4]. The first tool is a token-based approach used to detect *Duplicate Code*, in particular the type I[5]. The output of PMD denotes which snippets of a class also exists in other parts of the system. This output enables us to classify each clone according to their locality: i) **intra-class**, the clones of a class occur only inside the class itself; ii) **inter-classes**, a snippet of code classified as clone occurs only between different classes; iii) **mix-classes**, refers to clones occurring inter- and intra-class simultaneously. We use this tool because it enables us to classify the locality of clones and because Roy et al. [18] reports an extensive comparison of clone detection tools and they reported that PMD is good at detecting identical clones. Moreover, the tool DECOR identifies smells using detection rules based on the values of internal quality metrics[6]. In our paper, this tool is used identify the classes with the smells *Large Class* and *Complex Class*. The choice of using DECOR is driven by the fact that (i) it is a state-of-the-art smell detector having a high accuracy in detecting smells [12]; and (ii) it applies simple detection rules that allow it to be very efficient.

Thus, for each class of the analyzed systems, we identify whether the smells *Large Class*, *Complex Class* and/or *Duplicate Code* are present. For smelly classes, we also collect the numerical values of metrics used to identify each smell, e.g., for *Complex Class* we collect the *Cyclomatic Complexity* (McCabe) and for *Large Class* the metric NMD (number of methods declared) and NAD (Number of attributes declared), and for *Duplicate Code* the number of tokens

---

[3]https://pmd.github.io/

[4]*DEtection & CORrection* — https://bitbucket.org/ptidejteam/

[5]*Code are identical except for variations in whitespace, layout and comments* [3].

[6]An example of detection rule exploited to identify *Blob* classes can be found at [12].

of clone. These values are used to classify the intensity of smells (see Equation 1).

## 3.4 Analysis Method

We build, for each object system and for each research question, logistic regression models [9] relating a (dichotomous) dependent variable - indicating the presence or absence of clones - with independent variables represented by the quality indicators (code smells and their intensities).

Logistic regression assumptions relate to linearity, independence of errors, multicollinearity that are assumptions to consider. Considering all models of our RQs, linearity is not violated, because they use categorical predictors [5]. Considering that the classes analyzed in our study are independent and unrelated, we can assume the independence of errors. Multicollinearity is not violated, because the models use only one categorical predictor. Regarding the data, for any pair of categorical variables, it is useful to set up a contingency table to show the cell frequencies. We need to check if there are not more than 20% of cells with frequencies less than five, and any of the frequencies equals zero. [5].

For each model we analyze the Odds Ratio (OR) (Schumacker [20]) which is given by $e^{\beta_i}$. The higher the OR for an independent variable, the higher its ability to explain the dependent variable. The interpretation of the OR for the model built using quality metrics (smells), changes between the kinds of models due to the different aims of each research question. In general, for our research questions, the OR for an independent variable indicates the increment of chances for a class to be subject of clones in consequent of a one-unit increase of the independent variable.

## 4 EMPIRICAL STUDY RESULTS

This section discusses the results of our study, aimed at addressing the research questions. As explained, we performed a analysis of the assumptions of each logistic regression model. The underlying data used in this study is open and available [22].

## 4.1 The Number of Smells as a Factor

This part of our study is related to the research questions that investigate the prevalence of clones in the (co-)occurrence of the specific smells *Large* and *Complex Class*, not taking the intensity of these smells into account.

**RQ1.1:** *Is there a difference in the prevalence of clones comparing the entities where only CC occur and those entities where CC and LC co-occur? If so, how much this difference is associated to the smells CC and/or LC?*

**Table 2: Logistic Regression - RQ1.1.**

| Project | $\beta_0$ (SE) | $\beta_1$ (SE) | AIC | $R^2$ | IC for $\beta_1$ | | | Significance of Predictor |
|---|---|---|---|---|---|---|---|---|
| | | | | | Lower (5%) | OR | Upper (95%) | |
| ArgoUML | -0.383♦ (0.33) | -0.297♦ (0.38) | 205.91 | 0.003 | 0.394 | 0.742 | 1.415 | $\chi^2(1)$=0.59, $p$=0.444 |
| Cassandra | -1.504▽ (0.39) | -0.127♦ (0.46) | 143.77 | 0.001 | 0.415 | 0.880 | 1.962 | $\chi^2(1)$=0.07, $p$=0.787 |
| Lucene | -0.364△ (0.20) | 0.238♦ (0.25) | 360.55 | 0.002 | 0.832 | 1.269 | 1.944 | $\chi^2(1)$=0.86, $p$=0.353 |
| Hadoop | -1.992◄ (0.61) | 0.893♦ (0.67) | 98.82 | 0.021 | 0.873 | 2.444 | 8.496 | $\chi^2(1)$=2.00, $p$=0.157 |
| Ant | -1.504▽ (0.45) | 0.291♦ (0.51) | 138.64 | 0.002 | 0.595 | 1.337 | 3.274 | $\chi^2(1)$=0.33, $p$=0.564 |

Significance of Coefficients (R®): $0.001^\triangledown$; $0.01^\blacktriangleleft$; $0.05^\blacktriangleright$; $0.1^\triangle$; $1^\blacklozenge$. $^\otimes$Significant Predictor: $p < \alpha$.

This research question aims at comparing the prevalence of clones between the classes that *Complex Class* does not co-occur with *Large Class* and those which these smells co-occur. The model to answer this research question has a predictor variable and this is a categorical variable that describes two categories of smell *Complex Class*. Before building the logistic regression model, we performed a assumptions analysis of statistical test. In particular, we verify the possible frequency problems related to the dataset. This analysis reveals that the dataset of all projects satisfy those requirements.

The next step of our analysis verifies the significance of the predictor variable (see Table 2). For this research question, the *p*-value of $\chi^2$ test of all projects is not less than 0.10. Thus, we can not reject the null hypothesis that the predictor variable and the prevalence of clones (dependent variable) are independent. Therefore, for these software samples, there is no statistically significant evidence that the prevalence of clones in classes that exhibit only the smell *Complex Class* is different from the prevalence of clones occurring in the classes where the smells *Large Class* and *Complex Class* co-occur.

**RQ1.2:** *Is there a difference in the prevalence of clones comparing the entities where LC and CC co-occur and those entities where only LC occur? If so, how much this behavior is associated to the smells LC and/or CC?*

**Table 3: Logistic Regression - RQ1.2.**

| Project | $\beta_0$ (SE) | $\beta_1$ (SE) | AIC | $R^2$ | IC for $\beta_1$ | | | Significance of Predictor |
|---|---|---|---|---|---|---|---|---|
| | | | | | Lower (5%) | OR | Upper (95%) | |
| ArgoUML | -0.485△ (0.25) | -0.195♦ (0.32) | 239.68 | 0.002 | 0.483 | 0.822 | 1.407 | $\chi^2(1)$=0.36, $p$=0.548 |
| Cassandra | -1.887▽ (0.47) | 0.255♦ (0.54) | 131.64 | 0.002 | 0.550 | 1.291 | 3.387 | $\chi^2(1)$=0.23, $p$=0.633 |
| Lucene | -0.452♦ (0.27) | 0.326♦ (0.32) | 297.35 | 0.004 | 0.821 | 1.386 | 2.367 | $\chi^2(1)$=1.05, $p$=0.306 |
| Hadoop | -1.435◄ (0.49) | 0.336♦ (0.57) | 105.93 | 0.004 | 0.568 | 1.400 | 3.807 | $\chi^2(1)$=0.36, $p$=0.549 |
| Ant | $7.1E-15$♦ (0.63) | -1.213△ (0.67) | 121.21 | 0.026 | 0.095 | 0.297 | 0.922 | $\chi^2(1)$=3.09, $p$=0.079$^\otimes$ |

Significance of Coefficients (R®): $0.001^\triangledown$; $0.01^\blacktriangleleft$; $0.05^\blacktriangleright$; $0.1^\triangle$; $1^\blacklozenge$. $^\otimes$Significant Predictor: $p < \alpha$.

This research question is similar to the previous (RQ1.1), however, we investigate the prevalence of clones between the classes that

*Large Class* does not co-occur with *Complex Class* and those which these smells co-occur.

Based on our protocol of analysis, for each project, we observed that this dataset has neither assumption nor frequency problems (see Subsection 3.4). Thus, we can build the logistic regression model. Analyzing the significance of predictor of models (see Table 3), we observe that only for the Ant project the predictor variable and the prevalence of clones are associated. Thus, for other projects, the dependent and independent variables are not associated.

Considering only the logistic regression model of Ant project, we note that the coefficient $\beta_1$ also is significant and that the OR is less than one (0.29). This indicates that the prevalence of clones is more related to the isolated occurrence of smell *Large Class* than the co-occurrence of smells *Large Class* and *Complex Class*. This means that when we compare the chances of clones between classes that have the isolated occurrence of smell *Large Class* and those which the smells *Large Class* and *Complex Class* co-occur, the isolated occurrence of smell *Large Class* increase in 29% the chances of the class being involved with smell *Duplicate Code*. However, for this case, only 2.6% of the variability of the data can be explained by the model. Thus, this indicates that there are other factors that help to explain the prevalence of clones.

## 4.2 The Intensity of Smells as a Factor

We investigate what happens to the prevalence of clones when the intensity of the smells *Large* and/or *Complex Classe* become more critical.

**RQ2.1:** *Is there any association between the prevalence of clones and the intensity of smell* Complex Class*?*

**Table 4: Logistic Regression - RQ2.1.**

| Project | $\beta_0$ (SE) | $\beta_1$ (SE) | AIC | $R^2$ | IC for $\beta_1$ | | | Significance of Predictor |
|---|---|---|---|---|---|---|---|---|
| | | | | | Lower (5%) | OR | Upper (95%) | |
| ArgoUML | -0.810► (0.42) | 1.370► (0.75) | 50.51 | 0.069 | 1.166 | 3.937 | 14.512 | $\chi^2(1)$=3.44, $p$=0.064$^\otimes$ |
| Cassandra | -1.609▽ (0.44) | 0.510♦ (0.93) | 45.44 | 0.007 | 0.307 | 1.666 | 7.251 | $\chi^2(1)$=0.29, $p$=0.592 |
| Lucene | -0.465△ (0.24) | 0.331♦ (0.44) | 138.81 | 0.004 | 0.672 | 1.393 | 2.882 | $\chi^2(1)$=0.57, $p$=0.452 |

Significance of Coefficients (R®): $0.001^\triangledown$; $0.01^\blacktriangleleft$; $0.05^\blacktriangleright$; $0.1^\triangle$; $1^\blacklozenge$. $^\otimes$Significant Predictor: $p < \alpha$.

This research question investigates the prevalence of clones between the classes where the smell *Complex Class* occur at the low level of intensity and those that have this smell occurring at the high level of intensity. Observe that for this RQ, we do not consider the classes that have the co-occurrence of smells *Large Class* and *Complex Class*, which are investigated in RQ2.3.

In order to answer this RQ, firstly we checked the assumptions of logistic regression. Considering the contingency table of all projects, we observe that most part (77%) of classes with smell *Complex Class*$^{Low}$ does not have clones. Calculating the expected frequencies, we observe that the Hadoop and Ant project do not satisfy the necessary conditions of the statistical test. Thus, these projects will not be analyzed.

Table 4 denotes the output of logistic regression models for three projects. We observe that only for ArgoUML project, we reject the null hypothesis that the predictor variable and the prevalence of clones (dependent variable) are independent. Additionally, the significance of coefficient $\beta_1$ to the model of ArgoUML also is significant ($p-$value $< \alpha$). Thus, the value of Odds Ratio indicates the chances of the prevalence of clones to be associated with the intensity of smell *Complex Class*. In particular, the OR denotes that the classes with smell *Complex Class*$^{High}$ have 3.93 times more chance to participate in clones than classes that have the smell *Complex Class*$^{Low}$. However, for this case, only 6.9% of the variability of the data can be explained by the model. This means that in addition to the intensity of this smell there are another factor(s) that help to explain the prevalence of clones.

**RQ2.2:** *Is there any association between the prevalence of clone and the intensity of smell* Large Class?

Analogously to the RQ2.1, we analyze the prevalence of clones into the class that have the smell *Large Class*. Specifically, we investigate what happens with the prevalence of clones when the class have a low level of intensity of smell *Large Class* or when the intensity of this smell is high.

The first step before building the logistic regression models is verify the numerical problems. Thus, we construct the contingency tables and calculate their expected frequencies. From contingency table, we observe that the most part (67%) of classes with the smell *Large Class*$^{Low}$ does not have clones. Analyzing the expected frequencies, we observed that three projects (Cassandra, Hadoop and Ant) violate the assumptions related to the numerical problems (see Subsection 3.4). Thus, these projects are not considered in the following analysis.

According to the Table 5, the chi-square test is not significant for any projects. In other words, we do not have any evidence that the predictor variable and prevalence of clones are dependent/related. Thus, considering the five projects, we can not found any evidence of the association between the prevalence of clones and the intensity of smell *Large class*. Observe that this analysis does not consider the co-occurrence of smells *Large class* with other smells. This will be considered on the next research question.

**Table 5: Logistic Regression - RQ2.2.**

| Project | $\beta_0$ (SE) | $\beta_1$ (SE) | AIC | $R^2$ | IC for $\beta_1$ | | | Significance of Predictor |
| | | | | | Lower (5%) | OR | Upper (95%) | |
|---|---|---|---|---|---|---|---|---|
| ArgoUML | -0.405$^\blacklozenge$ (0.27) | -0.693$^\blacklozenge$ (0.86) | 87.03 | 0.008 | 0.099 | 0.5 | 1.883 | $\chi^2(1)$=0.70, $p$=0.402 |
| Lucene | -0.485$^\blacklozenge$ (0.31) | 0.149$^\blacklozenge$ (0.66) | 76.12 | 0.001 | 0.375 | 1.16 | 3.456 | $\chi^2(1)$=0.05, $p$=0.823 |

Significance of Coefficients ($R^\circledR$): 0.001$^\triangledown$; 0.01$^\triangleleft$; 0.05$^\triangleright$; 0.1$^\triangle$; 1$^\blacklozenge$. $^\otimes$Significant Predictor: $p < \alpha$.

**RQ2.3:** *Is there any association between the prevalence of clone and the intensity and co-occurrence of smells* Large Class *and* Complex Class?

This research question investigates the relation of the prevalence of clones with respect to the co-occurrence of smells *Large Class* and

*Complex Class*. In particular, we examined whether the intensity of these smells can be associated to the presence of clones. This means that we can build two models for each project, i) the first we considering the intensity of smell *Large Class* and ii) the other taking into account the intensity of smell *Complex Class*. In order to choose the better model we should evaluate each one of them.

As the first step, in order to obtain the expected frequencies, we construct the contingency table of each model for each project. For the model based on the intensity of smell *Complex Class*, all projects satisfy the assumptions related to the numerical problems and we also observed that the most part of clones occurs when the class has the smell *Complex Class* at the *High* level of intensity. In other words, the models denoted by "$\star$" in Table 6, should be evaluated. We also had a similar result for the models based on the intensity of smell *Large Class* (represented by "$\oslash$" in Table 6). Thus, both models should be analyzed.

Next, we should choose the best model for our dataset. Thus we analyze the significance of predictors of all models. Whether two different models are significant for the same project, we use the Akaike Information Criterion (AIC) values. According to Field et al. [5], AIC is a measure of fit and can be used to deciding which of two models fits the data better. The lower the AIC, the better is the fit of the model. Table 6 shows three projects with predictors significant (ArgoUML, Cassandra and Lucene) but we have only one situation where two different models are significant. This occurs with Lucene project and the analysis of AIC reveals that the best model is those based on the intensity of smell *Complex Class*. Interesting to note that for ArgoUML and Cassandra, the best model also is those taking into account the intensity of smell *Complex Class*. Thus, as resulting of this step, we conclude that the best model for all significant projects is based on *Complex Class* intensity (represented by "$\star$" in Table 6) and the model denoted as "$\oslash$" in Table 6, was not relevant for any project.

**Table 6: Logistic Regression - RQ2.3.**

| Project | | $\beta_0$ (SE) | $\beta_1$ (SE) | AIC | $R^2$ | IC for $\beta_1$ | | | Significance of Predictor |
| | | | | | | Lower (5%) | OR | Upper (95%) | |
|---|---|---|---|---|---|---|---|---|---|
| ArgoUML | $\star$ | -1.178$^\triangledown$ (0.33) | 0.822$^\triangleright$ (0.41) | 151.79 | 0.027 | 1.169 | 2.275 | 4.561 | $\chi^2(1)$=4.16, $p$=0.041$^\otimes$ |
| | $\oslash$ | -0.356$^\blacklozenge$ (0.35) | -0.462$^\blacklozenge$ (0.42) | 154.75 | 0.008 | 0.315 | 0.629 | 1.265 | $\chi^2(1)$=1.20, $p$=0.274 |
| Cassandra | $\star$ | -2.970$^\triangledown$ (0.72) | 1.772$^\triangleright$ (0.77) | 94.715 | 0.075 | 1.886 | 5.886 | 26.881 | $\chi^2(1)$=7.33, $p$=0.007$^\otimes$ |
| | $\oslash$ | -2.197$^\triangledown$ (0.52) | 0.810$^\blacklozenge$ (0.60) | 100.06 | 0.02 | 0.877 | 2.25 | 6.663 | $\chi^2(1)$=1.98, $p$=0.159 |
| Lucene | $\star$ | -1.321$^\triangledown$ (0.32) | 1.776$^\triangledown$ (0.38) | 200.28 | 0.113 | 3.206 | 5.906 | 11.35 | $\chi^2(1)$=24.90, $p$=0.000$^\otimes$ |
| | $\oslash$ | -0.510$^\triangleright$ (0.25) | 0.636$^\triangle$ (0.32) | 221.39 | 0.017 | 1.103 | 1.888 | 3.267 | $\chi^2(1)$=3.79, $p$=0.051$^\otimes$ |
| Hadoop | $\star$ | -1.658$^\triangleleft$ (0.54) | 0.822$^\blacklozenge$ (0.63) | 78.686 | 0.023 | 0.834 | 2.275 | 7.043 | $\chi^2(1)$=1.79, $p$=0.181 |
| | $\oslash$ | -1.609$^\triangleleft$ (0.54) | 0.740$^\blacklozenge$ (0.63) | 79.04 | 0.019 | 0.767 | 2.096 | 6.499 | $\chi^2(1)$=1.44, $p$=0.230 |
| Ant | $\star$ | -1.312$^\triangleleft$ (0.42) | 0.149$^\blacklozenge$ (0.51) | 107.26 | 0.001 | 0.503 | 1.16 | 2.814 | $\chi^2(1)$=0.08, $p$=0.773 |
| | $\oslash$ | -1.550$^\triangledown$ (0.41) | 0.545$^\blacklozenge$ (0.51) | 106.18 | 0.011 | 0.755 | 1.72 | 4.158 | $\chi^2(1)$=1.16, $p$=0.281 |

Significance of Coefficients ($R^\circledR$): 0.001$^\triangledown$; 0.01$^\triangleleft$; 0.05$^\triangleright$; 0.1$^\triangle$; 1$^\blacklozenge$. $^\otimes$Significant Predictor: $p < \alpha$.
$\oslash$ Model considering the intensity of the smell LARGE CLASS ($Y'$).
$\star$ Model based on the intensity of the smell COMPLEX CLASS ($Y''$).

Therefore, we take into account only the projects ArgoUML, Cassandra, Lucene, and their models marked with "★" in Table 6. Odds Ratio varies between 2.2 and 5.9, indicating that when smells *Large Class* and *Complex Class* co-occur in a class, the chances of that class has clone is 2.2—5.9 times larger in classes that has *Complex Class*$^{High}$ than classes with *Complex Class*$^{Low}$. However, only 2.7%—11.3% of the variability of the data can be explained by the model, indicating that there are other factors that help to explain the prevalence of clones and these considerations are consistent with those presented in the previous research questions

## 5 DISCUSSION

This section provides a qualitative perspective of our results. In particular, we discuss the statistically significant models in terms of their coefficients and predictors. For each one of these models, we examine the classes of projects that exhibit fragments of clones and have at least one of the smells of interest (*Large Class* and/or *Complex Class*). For instance, in the qualitative analysis of RQ2.1, we inspect the Java classes of ArgoUML that have clones and the smell *Large Class*. In this case, we inspect only ArgoUML project because the model of this project is the only statistically significant.

### 5.1 The Number of Smells as a Factor

This part of the discussion is related to the analysis of smells considering only their frequencies, namely RQ1.1 and RQ1.2.

**RQ1.1**. We studied the prevalence of clones in classes classified as *Complex Class* and they are organized into two groups according to their smells (only CC and co-occurrence of LC/CC). The results, for all projects, showed that the single occurrence of smell CC is not significant to the prevalence of clones. This observation is also valid for classes that have the co-occurrence of smells LC and CC. Moreover, the frequency of classes with co-occurrences of LC/CC is more than twice (×2.31), in average, the occurrence of CC only, indicating that although *CC* tends also to be *LC*, the fact of being large does not increase their chance to have clones.

**RQ1.2**. Its similar to the previous but the smell *Large Class* is used to organize the classes into groups. Statistical analysis has shown that the prevalence of clones is more associated to the single occurrence of smell *Large Class* than the co-occurrence of *Large & Complex Class* ($\beta_1 = -1,213, OR = 0,29$). However, this observation is valid only for Ant project and we observe that classes with LC and CC are almost the triple (×2.9 in average) than LC-only classes. So, we observe that the fact of being large does not increase their chance to have clones, and in Ant was the inverse. Thus, the results are project-specific. Moreover, to understand why Ant had LC-only classes with higher chances to have clones than LC-CC classes, we performed a qualitative analysis in the 10 classes classified as LC-only, where 50% of them also have the smell *Duplicate Code*. From these classes with clones, we observed that the 80% are "derivations" of the same entity, e.g., CCMklbtype has defined different attributes from those defined inside the class CCMkattr, but these classes shares: i) fully cloned methods (e.g., getComment-Command, getVersionCommand); ii) partially cloned methods (e.g., execute, checkOptions) and iii) other methods (ex. getVOB, get-TypeValueCommand). This is an indication that these clones are

practically the same, and they are distributed across several entities, suggesting that these classes provide little variability of data.

These clones could be avoided if adequate object-oriented practices (design patterns) had been adopted within those entities. Analyzing commits in these classes, we have observed that changes applied in one of these classes are commonly applied to others, where adaptations are minimal, reinforcing that object-oriented principles were not adequately followed. In the end, this finding could be helpful to indicate that when the system has high number of larges classes with high prevalence of clones those classes would deserve special attention.

### 5.2 The Intensity of Smells as a Factor

This subsection discuss qualitatively the results on the association of prevalence clones and the smells intensity of *Large Class* and *Complex Class* (RQ2.1, RQ2.2 and RQ2.3).

**RQ2.1**. The statistical results revealed that from the three analyzed projects (ArgoUML, Cassandra and Lucene), only one (ArgoUML) provide evidence for the association between the prevalence of clones and the intensity of *Complex Class*. In this case, we observed that classes with *Complex Class*$^{High}$ are more clone-prone than other smelly classes affected by *Complex Class*$^{Low}$. Specifically, the measure OR reveals that the chances are 3.93 times bigger. For this project, we have 15 classes with clones, where eight are classified as *Complex Class*$^{Low}$ and the rest as *Complex Class*$^{High}$. We examined these 15 classes, and report the main observations.

```
1  . . .
2  @Override
3  public boolean  stillValid (ToDoItem i,  Designer dsgr)  {
4    if  (! isActive ())  {
5      return  false ;
6    }
7    ListSet  offs  = i . getOffenders () ;
8    Object  f  = offs . get (0) ;
9    if  (! predicate (f ,  dsgr))  {
10     return  false ;
11   }
12   ListSet  newOffs = computeOffenders(f) ;
13   boolean  res  = offs . equals (newOffs);
14   return  res ;
15 }
```

**Code 1:** Clone with low complexity (*CrUnconventionalAttrName* – ArgoUML).

Analyzing clone fragments from CC$^{Low}$ classes, most of them (91%) are simple and/or small clones. For example, the class CrUnconventionalAttrName have three simple clone fragments, and one of them is partially in Code 1. This clone refers to a low complexity functionality. For clones CC$^{Low}$, we still observe that the used code template may influence the size of clones. Other examples exception handling and/or undesired condition with similar handling. This is highly prevalent in clones having nested *throw/try/catch*.

On the other side, clone fragments of CC$^{High}$ classes have shape, size and/or complexity different from those in CC$^{Low}$ classes. In CC$^{High}$ classes, we found no clones with complete functionality as the one in Code 1. In these classes clones are smaller fragments in a large sized functionality. For example, the method parseAttribute in the class AttributeNotationUml has 229 lines (*Long Method*), and has internally two different clones. Moreover, clones in CC$^{High}$

classes also correlate with the class itself having more control flow structures, i.e., are more complex too.

The average size of clones in $CC^{High}$ classes is larger compared to size of clones in $CC^{Low}$, and they are concentrated in the longer methods.

From the qualitative analysis, we conclude that for ArgoUML, the high complexity of a class is associated with the shape, size and complexity of the clones, corroborating with the statistical result. **RQ2.2**. This analysis is focused only on LC classes which are not also CC. Statistical analysis has shown that the intensity of LC in these classes is not a relevant factor to explain the prevalence of clones. For all systems, there are just a few (12.6%) $LC^{High}$ classes, and most of them are $LC^{Low}$. Because one of the classes is highly dominant against the other, no significant effect could be observed. **RQ2.3**. In this question, we consider only entities where LC and CC co-occur. Statistical models were significant for ArgoUML, Cassandra and Lucene, which are used in this discussion. Odds ratio indicated chance of clones being found in $CC^{High}$ to be 2.2-5.9 higher than being found in $CC^{Low}$.

In this part, we investigated how those clones found in ArgoUML, Cassandra and Lucene, could be classified over the type intra-class, inter-class ou mixed (both intra and inter). We manually analyzed each clone fragment of classes that manifest one or more LC/CC.

In ArgoUML, from classes with clones, 15 classes have CC-only, 24 have LC-only and 40 have both CC and LC co-occurring. The clones of classes CC-only or LC-only are almost all inter-class clones (95.8%). The clones of classes where CC and LC co-occur have also a high rate of inter-class clones (72.3%). We also analyzed Cassandra clones, which were similar to ArgoUML. In Lucene, inter-class clones in classes CC-only or LC-only correspond to 70.9% of clones and clones of classes where CC and LC co-occur have an even smaller rate of inter-class clones (36.2%), indicating that there are inherent project-specific design factors that induce the type of cloning (inter or intra) developers adopt.

We have observed that mixed clones (intra and inter class) have a very low frequency. The analysis of this RQ has shown that (i) clones occurring in entities having smells CC and LC co-occurring tend to be localized in entities with that same characteristic; (ii) the isolated occurrence of LC or CC does not prevent inter-class clones; (iii) the typical type of clone (intra or inter-class) seems to be related to project-specific decisions. These observations are valid only for systems (ArgoUML, Cassandra, Lucene) where the association is significant.

## 5.3 Practical Implications

The results for RQ2.3 have shown that clone fragments of large and complex classes are mostly restricted to that kind of class, i.e., clones occurring in classes where smells LC and CC co-occur are typical clones confined in other classes where smells LC and CC also co-occur. In other words, it is not likely that clones occur, at the same time, in large and complex classes, and also in a class that is not large and complex. This finding could be used to optimize the detection of *Duplicate Code*, that is, the detection algorithm could prune the search space for finding clones, in the sense, that large and complex class could be clustered to define a search space to look for clones.

Another practical implication is related to refactoring planning. As we have discussed, a reasonable part of clone fragments that occur in classes with just one of the smells (CC or LC) are inter-class clones. Also, inter-class clones are also highly prevalent in CC-LC classes, except for Lucene (36.2%). The implication is that removing this kind of clone would require a more sophisticated operation, e.g., the application of *Extract Superclass* or *Extract Class*, and thus making this process more risky and laborious. On the other hand, intra-class clones would be simpler to be removed applying the *Extract Method* refactoring. So, this kind of refactoring, in general, would be easier to be applied because it is local to just one class, i.e., it does not require investigating multiple classes. The implication is that this kind of refactoring could be applied more quickly, and could also be assigned for less experienced members of the development team, or those members of the team that do not have a broader knowledge of the system. However, this is just a hypothesis that should be further confirmed by other empirical study with human subjects.

## 6 THREATS TO VALIDITY

This section discusses the threats that could affect the validity of our study. Some threats should be considered in the analysis of the presented results. In this context, the internal validity examines whether the findings of an investigation are aligned with the study population. On the other hand, external validity refers to the extent to which research results can be generalized to other conditions.

One of the threats to external validity occurs because our results can not be generalized to other object-oriented programming languages, it because all the analyzed projects were developed in Java. To generalize this study, it is necessary to evaluate projects from other languages that also use the object-oriented paradigm. Furthermore, our empirical study only considers *open source* projects and the most part of them (80%) are maintained by *Apache Foundation*. Another threat is related to the sample size (five projects of software). However, the sample of three or more software is usual in exploratory studies. This type of study is characterized by the production of preliminar evidence that would support more comprehensive and extensive studies.

With respect to the internal validity, the analysis in this study does not include test classes (e.g., classes related to the *JUnit* test cases). These classes have different characteristics from those used in production (e.g., a preliminary analysis of our data revealed that they have larger clones in terms of LOC). We think that developers and/or researchers are more interested in the occurrence of smells on the production classes than the occurrence of smells in test classes. For classes of test cases, we found specifics studies [1, 25]. In general, our sample is aligned with the interests of the community.

The process of detecting the smells also is a threat to internal validity. According to de Mello et al. [4] performing ad-hoc manual identification of smells does not assure more effective results, so manually validating the detected smells would still not eliminate the threat. They observed that different context factors may influence on the conclusion about the incidence of a code smell. These factors are addressed to human aspects, such as the interaction among individuals and their professional roles. Moreover, DECOR is a state-of-the-art smell detector having a high accuracy in detecting

smells, according to Moha et al. [12] this tool has up to 88% of *Precision* and 100% of *Recall*. PMD also is good to detecting identical clones [18].

Finally, another threat to the internal validity is related to the scope and type of clones. This study considers only the clones of type I that occurred in smelly classes (*Large Class* and/or *Complex Class*). Nevertheless, there are clones in classes that have not been classified with these smells and these clones may present in a totally different way from those that we analyzed. However, the focus of this study is investigate the interaction of clones of type I and the smells LC and/or CC. Therefore, the investigation of other types of clones and/or the possible occurrence of them with other smells or classes without smells should be carried out in future works.

## 7 CONCLUSION

We investigated if the occurrence of smells *Large Class* and *Complex Class* could impact on the occurrence of clones. We considered different ways on how clones and these smells could interact: clones occurring in large-only classes, clones occurring in complex-only classes and clones occurring in classes that are simultaneously large and complex. Moreover, we investigate if the intensity of the LC and/or CC smells also plays a role in the prevalence of clones.

Our results have shown that conclusions are project-specific, i.e., they are not valid for all studied systems. In particular, only for Ant that the prevalence of clones is more associated to the co-occurrence of the smell *Large Class* and *Complex Class* than to the isolated occurrences of *Large Class* (see Table 3). So, the general hypothesis of prevalence of clones being associated with co-occurrence of LC and CC does not hold.

However, as we consider the intensity of smells in entities that have both the smells *Large Class* (LC) and *Complex Class* (CC), our data indicates that clone prevalence may be associated to the *High* intensity level of smell CC, either in CC only classes (ArgoUML), or CC co-occurring with LC (ArgoUML, Cassandra, Lucene), which in fact, may suggest a difference conclusion compared to that of Fowler and Beck [8] quote that "*...a class with too much code is prime breeding ground for duplicated code...*". We would prefer to say that a class with highly complex code, and with too much code, would be prime breeding ground for duplicated code. On the other side, independently on how our studies on smells have been carried out, they explain only a small part of clone occurrence, i.e., there are other factors that would help to explain the occurrence of clones in LC and/or CC classes.

We also observed that the class complexity and the intensity of the LC and CC smells influence some characteristics of the clone: shape (complete functionality or partial functionality clone), type (intra-inter clone), locality (clones restricted only to classes with specific configuration of smells), and size (LOC) of clones.

Future work includes i) replicating our study on proprietary systems, ii) enlarge the sample size, iii) investigate the influence of other smells on the prevalence of clones and iv) empirically investigate the practical implications.

## REFERENCES

[1] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley. 2012. An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In *IEEE International Conference on Software Maintenance*. , , 56–65.
[2] W.J. Brown, R.C. Malveau, H.W. McCormick, and T.J. Mowbray. 1998. *AntiPatterns: Refactoring Software, Architectures and Projects in Crisis*. Wiley, NY, USA.
[3] Debarshi Chatterji, Jeffrey C. Carver, and Nicholas A. Kraft. 2016. Code clones and developer behavior: results of two surveys of the clone research community. *Empirical Software Engineering* 21, 4 (2016), 1476–1508.
[4] R. de Mello, R. Oliveira, L. Sousa, and A. Garcia. 2017. Towards Effective Teams for the Identification of Code Smells. In *International Workshop on Cooperative and Human Aspects of Software Engineering*. IEEE, Piscataway, NJ, USA, 62–65.
[5] A. Field, J. Miles, and Z. Field. 2012. *Discovering Statistics Using R*. SAGE Pub., .
[6] F. A. Fontana, V. Ferme, and M. Zanoni. 2015. Towards Assessing Software Architecture Quality by Exploiting Code Smell Relations. In *International Workshop on Software Architecture and Metrics*. IEEE, Piscataway, NJ, USA, 1–7.
[7] F. A. Fontana, V. Ferme, M. Zanoni, and R. Roveda. 2015. Towards a prioritization of code debt: A code smell intensity Index. In *IEEE 7th International Workshop on Managing Technical Debt (MTD)*. , , 16–24.
[8] M. Fowler and K. Beck. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, .
[9] D.W. Hosmer, S. Lemeshow, and R.X. Sturdivant. 2013. *Applied Logistic Regression*. Wiley, .
[10] F. Jaafar, A. Lozano, Y. G. Gueheneuc, and K. Mens. 2017. On the Analysis of Co-Occurrence of Anti-Patterns and Clones. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. , , 274–284.
[11] D. Jiang, Peijun Ma, Xiaohong Su, and T. Wang. 2014. Distance Metric Based Divergent Change Bad Smell Detection and Refactoring Scheme Analysis. *Journal of Innovative Computing, Information and Control* 10, 4 (2014), 1519–1531.
[12] N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. Le Meur. 2010. DECOR: A Method for the Specification and Detection of Code and Design Smells. *IEEE Transactions on Software Engineering* 36, 1 (jan 2010), 20–36.
[13] W. Oizumi, A. Garcia, L. d. S. Sousa, B. Cafeo, and Y. Zhao. 2016. Code Anomalies Flock Together: Exploring Code Anomaly Agglomerations for Locating Design Problems. In *ACM International Conference on Software Engineering*. , , 440–451.
[14] W. Oizumi, L. Sousa, A. Garcia, R. Oliveira, A. Oliveira, O. I. A. B. Agbachi, and C. Lucena. 2017. Revealing Design Problems in Stinky Code: A Mixed-method Study. In *Brazilian Symposium on Software Components, Architectures and Reuse*. ACM, NY, USA, 1–10.
[15] F. Palomba, R. Oliveto, and A. De Lucia. 2017. Investigating code smell co-occurrences using association rule learning: A replicated study. In *IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation*. , , 8–13.
[16] Fabio Palomba, Marco Zanoni, Francesca Arcelli Fontana, Andrea De Lucia, and Rocco Oliveto. 2016. Smells Like Teen Spirit: Improving Bug Prediction Performance Using the Intensity of Code Smells. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. , , 244–255.
[17] Foyzur Rahman, Christian Bird, and Premkumar Devanbu. 2012. Clones: what is that smell? *Empirical Software Engineering* 17, 4-5 (2012), 503–530.
[18] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. 2009. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. *Sci. Comput. Program.* 74, 7 (May 2009), 470–495.
[19] Amanda Santana, Daniel Cruz, and Eduardo Figueiredo. 2021. An Exploratory Study on the Identification and Evaluation of Bad Smell Agglomerations. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC '21)*. ACM, New York, NY, USA, 1289–1297.
[20] R.E. Schumacker. 2014. *Learning Statistics Using R*. SAGE Publications, .
[21] Elder Vicente de Paulo Sobrinho, Andrea De Lucia, and Marcelo de Almeida Maia. 2021. A Systematic Literature Review on Bad Smells–5 W's: Which, When, What, Who, Where. *IEEE Transactions on Software Engineering* 47, 1 (2021), 17–66.
[22] Elder V. P. Sobrinho and Marcelo A. Maia. 2021. A Replication Package For The Paper "On the Interplay of Smells Large Class, Complex Class and Duplicate Code". https://doi.org/10.5281/zenodo.5103675
[23] D. Steidl and S. Eder. 2014. Prioritizing Maintainability Defects Based on Refactoring Recommendations. In *Proceedings of the International Conference on Program Comprehension (ICPC 2014)*. ACM, New York, NY, USA, 168–176.
[24] Michele Tufano, Fabio Palomba, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Andrea De Lucia, and Denys Poshyvanyk. 2017. When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away). *IEEE Transactions on Software Engineering* 43, 11 (2017), 1063–1088.
[25] Bart Van Rompaey, Bart Du Bois, Serge Demeyer, and Matthias Rieger. 2007. On The Detection of Test Smells: A Metrics-Based Approach for General Fixture and Eager Test. *IEEE Transactions on Software Engineering* 33, 12 (dec 2007), 800–817.
[26] Stephane Vaucher, Foutse Khomh, Naouel Moha, and Yann-Gael Gueheneuc. 2009. Tracking Design Smells: Lessons from a Study of God Classes. In *2009 16th Working Conference on Reverse Engineering*. IEEE, , 145–154.
[27] A. Yamashita and L. Moonen. 2013. Exploring the impact of inter-smell relations on software maintainability: An empirical study. In *2013 35th International Conference on Software Engineering (ICSE)*. , , 682–691.