An Exploratory Evaluation of Code Smell Agglomerations

Amanda Santana (amandads@dcc.ufmg.br)
Universidade Federal de Minas Gerais
Eduardo Figueiredo
Universidade Federal de Minas Gerais
Juliana Alves Pereira
Pontifical Catholic University of Rio de Janeiro
Alessandro Garcia
Pontifical Catholic University of Rio de Janeiro

Research Article

Keywords: code smell, code modularity, software quantitative analysis, metrics

Posted Date: October 27th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3486799/v1

Additional Declarations: No competing interests reported.

An Exploratory Evaluation of Code Smell Agglomerations

Amanda Santana[1*], Eduardo Figueiredo[1], Juliana Alves Pereira[2], Alessandro Garcia[2]

[1]Department of Computer Science, Federal University of Minas Gerais, Antonio Carlos Avenue, Belo Horizonte, 31270-901, Minas Gerais, Brazil.
[2]Informatics Department, Pontifical Catholic University of Rio de Janeiro, Marquês de São Vicente, Rio de Janeiro, 22451-900, Rio de Janeiro, Brazil.

*Corresponding author(s). E-mail(s): amandads@dcc.ufmg.br;
Contributing authors: figueiredo@dcc.ufmg.br; juliana@inf.puc-rio.br; afgarcia@inf.puc-rio.br;

Abstract

Context. Code smell is a symptom of decisions about the system design or code that may degrade its modularity. For example, they may indicate inheritance misuse, excessive coupling and size. When two or more code smells occur in the same snippet of code, they form a code smell agglomeration. **Objective.** Few studies evaluate how agglomerations may impact code modularity. In this work, we evaluate which aspects of modularity are being hindered by agglomerations. This way, we can support practitioners in improving their code, by refactoring the code involved with code smell agglomeration that was found as harmful to the system modularity. **Method.** We analyze agglomerations composed of four types of code smells: Large Class, Long Method, Feature Envy, and Refused Bequest. We then conduct a comparison study between 20 systems mined from the Qualita Corpus dataset with 10 systems mined from GitHub. In total, we analyzed 1,789 agglomerations in 30 software projects, from both repositories: Qualita Corpus and GitHub. We rely on frequent itemset mining and non-parametric hypothesis testing for our analysis. **Results.** Agglomerations formed by two or more Feature Envy smells have a significant frequency in the source code for both repositories. Agglomerations formed by different smell types impact the modularity more than classes with only one smell type and classes without smells. For some metrics, when Large Class appears alone, it has a significant and large impact when compared to classes that have two or more method-level smells of the same type.

Conclusion. We have identified which agglomerations are more frequent in the source code, and how they may impact the code modularity. Consequently, we provide supporting evidence of which agglomerations developers should refactor to improve the code modularity.

Keywords: code smell, code modularity, software quantitative analysis, metrics

1 Introduction

Code smells are symptoms of the decisions on the system design and code that may lead to degradation of its modularity and quality [1]. For instance, some smells are related to size, high coupling, high levels of complexity, low cohesion, and inheritance misuse. These problems make the maintenance and extension activities costly since the code is more complex, inflexible, harder to understand, and changes may cause ripple effects. A recent study [2] found that developers spend up to 58% of their time understanding the source code before performing maintenance tasks.

Code smells may affect quality properties of the system, such as understandability, extensibility, and reusability [3, 4], and they are mostly introduced at artifact creation, usually when the developer is under pressure [5]. To remove smells, developers need to refactor the smelly code. *Refactoring* is an activity in which modifications in the source code are made without changing its external behavior, usually to improve the code quality [1]. These changes may involve, for instance, the removal of code smells, and code simplification [6].

Several studies evaluated the impact of isolated code smells on software modularity, such as maintenance effort [4, 7, 8], their distribution along the systems [9, 10], software comprehension [11], error proneness and change proneness [8, 9, 12, 13, 14, 15, 16]. In addition, studies show that when two or more code smells co-occur in the same piece of code (e.g, class or method), forming a *code smell agglomeration*, they make the code even harder to maintain and to understand [17, 18, 19, 20]. Even though code smells in isolation have been extensively researched in the literature [21], agglomerations only recently started getting attention. However, most works focus on identifying the most frequent agglomerations in the source code [22, 23, 24, 25], and few studies tried to understand how they impact aspects of the source code modularity [9, 26] and the code life-cycle [27, 28, 29].

Furthermore, some of these studies rely on small datasets composed of two to four systems [19, 24, 28, 30, 31, 32]. Meanwhile, most of the studies that evaluate a large number of systems use the detection results from only one detection tool. These tools rely on specific metrics thresholds, thus different tools for the same smell do not meaningfully agree with their answers. Differently from previous work, we use multiple tools and consider the presence of agglomerations formed by the same smell type, such as the agglomerations formed by two or more Long Methods in the same class. These agglomerations are present due to our evaluation being at the class level, consequently, they can not appear at method level.

This paper expands and improves the analysis of our previous work [33]. Our main goal is to identify which agglomeration impacts negatively modularity considering the code complexity, coupling, and inheritance use. In our previous study, we evaluated the impact of agglomerations in twenty systems extracted from the Qualita Corpus [34]. We have also analyzed how agglomerations are spread along the systems. In this work, we improved the impact analysis using statistical testing combined with effect size calculation. We then mine a new dataset composed of ten different systems extracted from the GitHub platform. Since the systems from the previous study are from 2010, we aimed to verify if the results found are confirmed for systems that are actively being maintained by the community, that is, systems that had commits merged after 2021. Thus, for comparison reasons, we conducted an individual analysis of each dataset, and later compared their results in order to verify their agreement. In both datasets, we evaluate the following smells: Large Class (LC), Long Method (LM), Refused Bequest (RB), and Feature Envy (FE). We selected these smells considering (i) their different perspectives of code modularity degradation (see Section 3.7) and (ii) if they were extensively researched in the literature [21].

In summary, this work is divided into two studies: the Qualita Corpus and the GitHub studies. In both studies, we aim to identify which agglomerations are frequent on the systems and how they may impact the software modularity. For this purpose, we used frequency statistics [35] and frequent itemset mining [36], and we combine hypothesis testing [35] to verify how they may impact the aspects of modularity. Beyond that, we aim to understand if there are some smells that in combination with other smell types impact the most/the least the modularity aspects. This is interesting since aspects of modularity may be affected differently by different agglomerations. To evaluate the similarities and differences between both studies, we compared and raised possible divergences in their results. This analysis allows us to verify if the agglomerations are sensitive to the repository choice.

In the Qualita Corpus Study, we use a subset of 20 Java systems from a state-of-the-art dataset [34], trying to diversify the system characteristics, such as domain, size, and maturity. In total, we have found 15,690 smells for this dataset. In the GitHub Study, we selected 10 popular Java systems, and in total, we found 1,388 smells.

Our main findings for both datasets are:

- when analyzing the agglomerations at the class level, the agglomerations formed by two or more smells of the same type are highly present in the dataset (36.3% of all agglomerations for the Qualita Corpus dataset, and 33.8% for the GitHub). We also highlight that the most frequent agglomeration type is FE+FE (i.e. at least two Feature Envy in the same class;
- we have identified which agglomerations formed by different smells were more frequent in both datasets. They are LC+FE (Support of 0.307 and 0.352 for the Qualita Corpus and GitHub, respectively) and LM+FE (Support of 0.472 and 0.347 for Qualita Corpus and GitHub, respectively);
- for both datasets, we have found that agglomerations formed by two or more different smell types impact system complexity, cohesion and coupling the most. However, we could not find a statistical difference between the datasets for inheritance;

- for the GitHub dataset, agglomerations formed by two or more smell does not have a significant difference between their impact on the modularity. Meanwhile, for the Qualita Corpus, we have found differences in their impact, mostly on the class complexity;
- we also make our data available to encourage replications and further studies on agglomerations [37].

Audience. Researchers and practitioners in maintenance and evolution shall benefit from our experiments and insights. We show evidence of the agglomeration impact, and we expect that developers could use this information to evaluate which agglomerations to refactor and improve the code modularity. The remainder of this paper is structured as follows. Section 2 describes the main concepts used in this work. Section 3 presents our study design. Sections 4 and 5 discuss the findings obtained for the Qualita Corpus dataset and the GitHub dataset, respectively. Section 6 compares the results from both datasets and presents the implications for practitioners and researchers. Section 8 discusses related works. Section 7 considers threats to both studie's validity. Finally, Section 9 concludes our work and presents possibilities for future work.

2 Code Smell Agglomerations

2.1 Evaluated Code Smells

Code smells are symptoms of decisions made by developers on the source code that may degrade its modularity [1]. We opted to study four code smells: Large Class (LC), Long Method (LM), Feature Envy (FE), and Refused Bequest (RB). The rationale for this choice was the availability of at least three detection tools to detect each smell, ensuring that different detection strategies are being used in their detection. This strategy allows us to explore different approaches to what developers consider as a code smell. We also aim to expand the current state-of-the-art knowledge by exploring smells that were not extensively researched (FE and RB) combined with extensively researched smells (LC and LM) [21].

LC occurs when a class tries to do too much and it usually have high number of lines of code. LM occurs when methods have too many lines of code, frequently with long explanations of how it works. FE occurs when a method seems more interested in being in another class. RB occurs when a class does not use their parent's behavior. It is important to mention that we considered Large Class, Brain Class, and God Class as the same smell. We also considered the Large Method, Brain Method, and God Method as the same smell. This choice was motivated by their similar definitions [1, 38, 18], that are highly related to the size and complexity of the code.

2.2 Classification of Code Smell Agglomerations

A *code smell agglomeration* occurs when two or more code smells co-occur on the same module. In our paper, we work with agglomerations formed inside a class. We then classify the agglomerations according to the smell that it contains: Homogeneous and Heterogeneous. Moreover, we rely on two other definitions of whether a class has (has not) a code smell: Isolated Code Smell and Clean Class.

- *Homogeneous Agglomeration* occurs when a class has two or more code smells but of only one type. They are possible because we are considering a higher granularity, *i.e.,* classes instead of method agglomeration. To better understand their behavior, we further classify the Homogeneous Agglomerations according to the analyzed smells: Homogeneous FE and Homogeneous LM. As an example of a Homogeneous LM, consider the class *org.apache.naming.resources.ResourceAttributes*[1] from the Tomcat system, containing four LM (*getCreation*, *getCreationDate*, *getLastModified* and *getLastModifiedDate*). All of them are long and present a series of switch statements.
- *Heterogeneous Agglomeration* occurs when a class contains two or more code smells of different types. As an example of Heterogeneous Agglomeration, we consider the class *com.badlogic.gdx.graphics.g3d.particles.batches.-BillboardParticleBatch*[2] from the Libdgx system. This class presented both the LC and RB smells. This class is long, with more than 600 LOC, and beyond implementing the rules to personalize the particles, it also contains code that implements the rendering and manipulations, such as saving and loading. This class also presents an RB, because it only calls two times its superclass, and it overrides most of its superclass functionalities, adding new behavior to the class.
- *Isolated Code Smell* occurs when a class contains only one code smell. It can be further characterized according to the type of smell: Isolated FE, Isolated LM, Isolated LC, and Isolated RB.
- *Clean Class* defines a class with no code smell found.

We highlight that each class in our dataset is assigned to one label, according to the number of smells and their respective type. For instance, a class containing a Large Class and a Feature Envy will be assigned to the Heterogeneous label. Meanwhile, a class that presents only a Large Class is labeled as Isolated Large Class.

3 Study Design

3.1 Study Design Overview

Figure 1 presents an overview of this work design. The rounded squares present the main activities. The letters inside the circles represent the order in which each activity was conducted. The filled arrows show that the left item served as input for the next activity. First, we select systems to be evaluated. We selected 20 systems from our previous study [33], all from the Qualita Corpus repository (Section 3.3). For comparison, we selected 10 systems from GitHub repositories (Section 3.4). In Step A, we apply five detection tools to build our smell dataset (Section 3.5). We then calculate for each system the selected modularity metrics in Step B (Section 3.7).

In Step C, we identify and classify the Heterogeneous and Homogeneous Agglomerations for each dataset (Section 3.6). In Step D, we calculate how the agglomerations impact the system modularity (Section 3.7). Finally, Step E compares the results

[1] https://github.com/amandads/AgglomerationsExtensionProject/blob/main/Samples/ResourceAttributes.java
[2] https://github.com/amandads/AgglomerationsExtensionProject/blob/main/Samples/BillboardParticleBatch.java

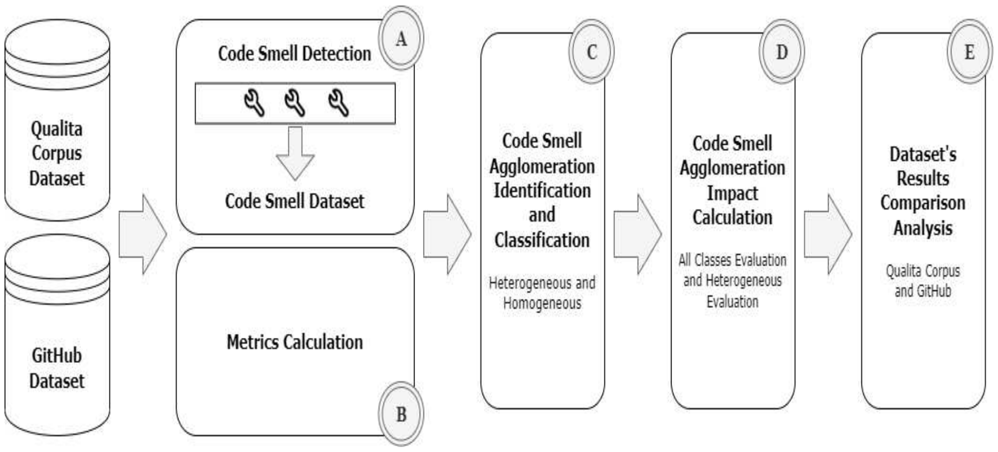
Qualita Corpus Dataset
GitHub Dataset
Code Smell Detection
Code Smell Dataset
Metrics Calculation
Code Smell Agglomeration Identification and Classification
Heterogeneous and Homogeneous
Code Smell Agglomeration Impact Calculation
All Classes Evaluation and Heterogeneous Evaluation
Dataset's Results Comparison Analysis
Qualita Corpus and GitHub
A
B
C
D
E

Fig. 1: Methodology steps

obtained in both datasets (Section 6) to answer the proposed research questions described in Section 3.2.

3.2 Research Questions

The goal of this paper is to identify which code smell agglomerations are more common on the source code and to explore their impact on software modularity. Moreover, beyond expanding the current knowledge on agglomerations, we provide evidence of their harmfulness considering different perspectives of good object-oriented software modularity. The following research questions (RQs) guided us in the data collection and analysis.

RQ1: *Which Homogeneous Agglomerations are more frequent in the source code?*

RQ2: *Which Heterogeneous Agglomerations are more frequent in the source code?*

RQ1 and RQ2 aim to provide evidence of the frequency of agglomerations and further investigate if they need more exploration. RQ1 provides evidence of the presence of Homogeneous Agglomerations, while RQ2 focuses on the presence of Heterogeneous Agglomerations.

RQ3: *How do code smell agglomerations impact the system modularity?* This RQ has two main goals. The first one is to understand how such agglomerations may impact different aspects of modularity. The second goal is to investigate if agglomerations have a more negative impact on modularity than Isolated and Clean Classes.

RQ4: *Do the different types of Heterogeneous Agglomerations have a uniform impact on the system modularity?* Since in RQ3 we aggregated all Heterogeneous Agglomeration in a single label to make the presentation of the results simpler, we aim with this RQ at understanding how different Heterogeneous Agglomerations behave. With this aim, we compare different types of Heterogeneous Agglomerations, investigating whether different combinations of smells have more or less impact on the modularity.

3.3 Qualita Corpus Dataset

In the first phase of our study, we opted to create a new code smell dataset, even though some were available in the literature [22, 25]. Our main motivation was that we had difficulties acquiring the dataset details, such as how the data was collected and how code smells were identified. To create our dataset, we selected a subset of 20 open-source Java systems from the Qualita Corpus [34], a curated collection of systems widely evaluated in the literature [21, 22]. We selected them according to: *(i)* use in industrial and open source projects; *(ii)* support in different development stages; and *(iii)* are often used in the code smell literature [21]. For instance, while HtmlUnit and JMeter support testing, JHotDraw can be used to model systems. These criteria allow us to achieve a more generalizable result since we choose systems with different sizes, purposes, development processes, and maturity.

Table 1 presents the selected systems from Qualita Corpus phase and their characteristics. The first column presents the system name and its version. The second to fourth columns present, respectively, the systems' lines of code (LOC), the number of classes, and the number of methods (NOM). The fifth column presents the system domain according to Tempero's work [34], and the last column presents the absolute number of code smells we found in the system. We highlight we have cleaned companion classes, such as test, demo, samples, examples, aspect-oriented, and android classes. Our focus in this work is on evaluating only classes that implement object-oriented code. We also highlight that the systems already have their dependency solved [39].

We can observe from Table 1 that the systems vary from small, such as Commons-Logging which has only 2.7 KLOC distributed in 28 classes and 302 methods, to large systems, such as Hibernate with 179.6 KLOC distributed in 3,264 classes and 25K methods. The number of code smells in each system varies greatly, ranging from 1 in SquirrelSQL to 1,699 in Weka. In total, we analyzed over 1.79M lines of code, distributed in 24,727 classes and 186,199 methods.

3.4 GitHub Dataset

The second phase of this work aims at verifying if the results obtained from the Qualita Corpus dataset hold for systems that are being actively maintained, such as the top-stared GitHub (GH) repositories. This allows us to verify how generalizable the results for code smells agglomerations are, since different practices and technologies are employed in both datasets.

For this purpose, we first searched on GitHub for Java repositories, and sorted them by the number of stars. The systems were obtained from April 12, 2021 to February 15, 2022, due to the process of compiling and using the detection tools. It is worth noticing that we used the newest version of the system available at the respective data collection date. The following exclusion criteria were used to remove some repositories. *(i)* repositories that only contained educational code; and *(ii)* repositories with less than 90% of its code written in Java. We restricted to Java systems, due to our infrastructure requiring the system to be compiled in the Eclipse IDE. The large use of libraries from other languages makes the activity of adjusting the systems to our infrastructure unfeasible, even with dependency managers, such as Maven and Gradle.

Table 1: Selected Systems from Qualita Corpus

Name	LOC	#Classes	NOM	Domain	#Smells
checkstyle-5.6	22,877	453	2,782	IDE	231
commons-codec	7,314	96	772	tool	4
commons-io	9,309	134	1,280	tool	7
commons-lang	28,167	269	3,223	tool	19
commons-logging	2,712	28	302	tool	3
hadoop-1.1.2	211,436	2,747	17,566	middleware	173
hibernate-4.2.0	176,607	3,264	25,062	database	148
htmlunit-2.8	37,135	545	5,040	testing	273
jasperreports-3.7.4	126,793	1,779	16,527	visualization	1,109
jfreechart-1.0.13	80,619	638	8,645	tool	540
jhotdraw-7.5.1	76,983	638	7,584	graphic	470
jmeter-2.5.1	70,961	955	7,613	testing	680
lucene-4.2.0	223,880	3,261	17,550	tool	524
quartz-1.8.3	22,609	248	2,697	middleware	186
spring-3.0.5	133,713	2,733	17,838	middleware	40
squirrelsql-3.1.2	6,011	169	689	database	1
struts-2.2.1	98,396	1,580	10,691	middleware	608
tapestry-5.1.0.5	37,565	1,395	5,739	middleware	307
tomcat-7.0.2	162,621	1,700	15,634	middleware	952
weka-3.6.9	254,947	2,095	18,965	tool	1,699
Total	1,790,655	24,727	186,149	-	7,974

First, if the system did not fit the exclusion criteria, we downloaded the system from the GitHub repository and tried to solve its dependencies automatically. If it aroused some errors, we tried to manually adjust the environment to compile it. We repeated this process until we got 10 systems. The list of systems is presented in Table 2, organized in a similar way to Table 1. The only difference is that in the seventh column, we present the number of stars at the system download.

Table 2: Selected Systems from GitHub

Name	LOC	#Classes	NOM	Domain	#Smells	Stars
DBeaver-21.0.2	348,609	6,449	36,575	database	705	19.7K
FastJSON-1.2.76	43,536	249	1,996	tool	66	23.2K
GSON	11,641	228	920	tool	26	19.4K
jsoup1-1.14.2	17,290	242	1,540	tool	19	9.3K
JUnit-4.13.2	10,769	310	1,541	testing	11	8.3K
libgdx-1.9.14	208,028	2,714	39,338	development	512	19.6K
NanoHTTPD-2.3.1	3,566	67	380	middleware	6	6.2K
Netty-1.7.18	5,217	138	712	middleware	10	28.5K
RetroFit-1.6.0	4,706	115	403	middleware	8	37.8K
WebMagic-0.7.3	5,857	163	819	tool	25	10.2K
Total	659,219	10,675	84,224	-	1,388	182.2K

From Table 2, we can observe that we got good coverage of domains, with systems having different domains. We also covered a variety of sizes, with DBeaver being the largest system, with 348.6 KLOC distributed on 6.4K classes and 36.6K methods. The smallest system was NanoHTTPD, with 3.6 KLOC distributed on 67 classes and 380 methods. Finally, the number of smells found on each system varies, raging 6 smells for the NanoHTTPD and 705 for DBeaver. In total, we have analyzed over 659 KLOC, 10,675 classes, and 84,224 methods.

3.5 Building the Dataset

Building a large dataset of code smells manually is an expensive and subjective activity, since the code and its relationships should be investigated. Furthermore, the developer's perspective about what they considered as smelly code may be influenced by different aspects, such as their system knowledge and experience [20, 40]. Several techniques for code smell detection have been proposed in the literature [41, 42, 43, 44, 45, 46, 47], employing different methods, such as metrics, machine learning solutions, historical data, and textual information. We opted to use automatic detection tools, since manually investigating the presence of smells is unfeasible due to the necessary effort to understand and evaluate the code of 30 systems.

For this purpose, we selected three detection tools for each of the analyzed smells, according to their availability and target smell. In cases where more than three tools are available to detect the smell, we opted to use those that were most used to detect such smell [21]. For instance, although Organic [26] also detects the LC smell, we opted to use JDeodorant, PMD, and JSpIRIT.

To build our ground dataset, we used a voting method [48], in which each instance (a class or method) received three votes from three different detection tools on the presence of a certain smell. Each vote represents if the tool detected the instance. If the instance received two or more positive votes, then the instance is added to our code smell dataset. This process is repeated for every smell evaluated. We choose this method due to empirical evidence that having more "thinking units" provides a more accurate prediction [36], leading to a more reliable dataset, even though it may miss some true positive smelly instances [49].

The smells were detected by the following tools. LC: JDeodorant, PMD and JSpIRIT. RB: JSpIRIT, DECOR, and Organic. FE: JDeodorant, JSpIRIT, and Organic. LM: JDeodorant, DECOR, and Organic. Beyond their availability, we selected these tools considering the detection strategy that they employs. For instance, DECOR uses lexical properties, structural properties and relationships to identify the smells [50]. Meanwhile, JDeodorant identifies code smells through the detection of refactoring opportunities [51]. The other tools use metrics and thresholds to detect the smells, however, none of the analyzed metrics in this work are part of their detection strategy [26, 52, 53].

3.6 Identifying Homogeneous and Heterogeneous Agglomerations

Identifying agglomeration is not a simple task, since all combinations of code smells must be considered, and not all of them are meaningful [36, 54]. To identify the Heterogeneous Agglomerations, we used frequent itemset mining [36]. This technique is powerful, due to its capacity to explain the relationship between the different smells present in the sets. With its meaningfulness metrics, it can identify a significant group of items (smells) that appear together in the dataset [36], according to different measurements.

To identify these sets, we used the *Support* and *Confidence* measurements [36]. *Support* measures how frequently the combination of items appears together in the Heterogeneous dataset. *Confidence* measures how certain we are of the detected association. *Confidence* values below 0.5 are not interesting [36]. In this work, we consider all combinations in which the *Support* is above 0.15 and the *Confidence* is above 0.5. The *Support* thresholds were selected after experimenting with different values. We first checked thresholds used in the literature [22, 25, 33], for instance, 0.01 and 0.9, but we could not find meaningful results. This may be due to these thresholds being extreme values. For instance, in setting the Support to 0.01, we obtain all the itemsets that have a frequency of at least 1% in all datasets. As a result, we obtained all possible combinations of smells as meaningful. Meanwhile, a Support of 0.9 indicates that the itemset has to be present in more than 90% of our class instances. We used the Apriori Algorithm [55] to obtain the frequent itemsets in the R package *arules*[3]. However, this technique has some drawbacks: a certain smell S can only appear one time in the identified set. Consequently, we cannot use this technique to identify Homogeneous Agglomerations. We then opt to use frequency statistics and variance analysis to characterize their presence [35].

3.7 Modularity Metrics Impact

In previous studies [12, 56], it was identified that not all smells are harmful to the code quality. Even more, their harm depends on the context in which the smell is present [57]. For instance, Alkharabsheh et al. [49] have found that LC smells are more impacting in certain software domains. Our goal is to evaluate which agglomeration types harm the most code modularity. However, attributes of quality are difficult to estimate, since they depend on the context of the system and on who is evaluating. To address this complexity, we used source code metrics that capture characteristics of modularity as our proxy of quality. We selected the metrics considering *(i)* if they are not part of the detection strategy of the tools; *(ii)* they cover several aspects of modularity, such as complexity, coupling, cohesion, and inheritance use; and *(iii)* if there were studies that correlated the metric to the modularity attribute.

In this work, we explore two views of impact: a general one, in which we compare the agglomerations based on the types described in Section 2.2 (inspired by previous work [58]), allowing us to verify if agglomerations in general are more harmful than classes with one/no smell. In the second view, we expand the Heterogeneous Agglomeration analysis according to the smells that it contains and verify if there is any combination of smells that makes metric values worse when compared to other agglomerations. This information can be used to prioritize which Heterogeneous Agglomerations to refactor.

To achieve our goals, we first use the Mann Whitney U Test [35], a non-parametric statistical test, to evaluate if, for each pair of agglomerations, their distribution is

[3]https://cran.r-project.org/web/packages/arules/index.html

equal. We use this test since our data do not follow a normal distribution, are not paired, and can be ranked. For this test, we have used *scipy.stats* library in Python[4]. Be $i,j \in A$, with A = {Set of Agglomerations Types} and $i \neq j$. Be $m \in$ M = {Set of Evaluated Metrics}. We have the following hypothesis.

H_{0-mij} = Distribution of metric m for categories i and j are equal.
H_{1-mij} = Distribution of metric m for categories i and j are not equal.

Since we are comparing multiple categories, we use the Bonferroni Correction [35] to correctly interpret the obtained p-values. To complement the statistical analysis, we use Cliff's Delta measure of effect size in order to provide information on how different the medians are. We highlight that we are not investigating if, for a metric m, an agglomeration surpasses a specified threshold. We are interested in investigating if two categories have different distributions for metric m, and in case this is true, we want to understand how much they differ. This difference is provided by the Cliff's Delta. It is important to notice that effects can be positive or negative. For example, suppose that the comparison between categories X-Y had a positive effect. This indicates that, usually, the metric value for the X is higher than Y. If we change the order to Y-X, the effect will be negative, and Y tends to have a lower metric value than X. The Cliff's Delta was calculated using the Python library[5]

With this statistical framework, we can provide solid evidence of whether agglomerations impact modularity aspects. That is, if they typically have higher metrics values than isolated and clean classes. Table 3 presents the selected metrics, the definition used, and the aspect of modularity they represent. We highlight that the selected metrics had their correlation to the modularity previously explored by the literature [18, 59, 60]. We used the CK tool [61] to calculate the metrics in Table 3.

Table 3: Metrics and its Modularity Aspects

Metric	Definition	Aspect
Coupling Between Objects (CBO)	This metric counts the number of dependencies a class C has, for instance, method calls. [59]	Coupling
McCabe's Complexity (McCabe)	It counts the number of conditionals and loop instructions. [62]	Complexity
Maximum Nesting (MaxNest)	It calculates the highest number of blocks nested together in class C. [60]	Complexity
Depth of Inheritance Tree (DIT)	It measures how deep the class is in the inheritance tree. [59]	Inheritance
Number of Children (NOC)	This metric counts how many direct children a class C has. [59]	Inheritance
Lack of Class Cohesion (LCOM)	We used a normalized version of the original LCOM, that calculates how much of the methods use the same attributes in their body. The lowest the value, the highest the cohesion of the class. [63, 59]	Cohesion

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html
[5] https://github.com/neilernst/cliffsDelta.

These metrics approach different views of modularity. High coupling means that the elements in the source code have a strong dependency on each other. This high level of dependency may hinder the maintenance activities due to the comprehension and possible ripple-effects of such relationships. High complexity makes the source code harder to understand and maintain. A lack of cohesion indicates that the class has different reasons to change. That is, it has too many responsibilities. Finally, in the inheritance aspect, we want to verify if classes that have smells related to inheritance are more prone to be lower on the hierarchy (high DIT) and have too many children.

4 Qualita Corpus Results

4.1 Overview of Smells Found

Table 4 presents in each line the number of smells found. The first column shows the smell name. The second column shows the absolute number of smells identified by the voting method (see Section 3.5). The third column presents the proportion of code smell considering the total number of smells. The last column presents the ratio of smelly classes/methods in our dataset. The denominator in this column varies depending on the smell granularity (class or method).

Table 4: Code Smell Found on Qualita Corpus

Code Smell	**#Smell**	**#Smell/#TotalSmell**	**% Smelly Elements**
Large Class (LC)	580	7.27%	2.36%
Refused Bequest (RB)	966	12.11%	3.91%
Feature Envy (FE)	5,128	64.31%	2.75%
Long Method (LM)	1,300	16.31%	0.7%
Total	7,974	100%	-

We can observe in Table 4 that about 19% of the smells are at class level (i.e., LC and RB) and 81% are at method level (i.e., FE and LM). Higher participation of smells at the method level is expected since a system has more methods than classes. We can point out that 64% of the code smells found are FE. However, observing the relative number of affected classes and methods (fourth column), we can verify that the most frequent smell with respect to the total elements (i.e., classes or methods) is RB, with 3.91% of all classes being affected by this smell. Moreover, Table 4 provides us motivation to discuss how the imbalance in the smell distribution may bias our results (mainly discussed in Section 6).

4.2 Exploring Homogeneous Agglomerations

Table 5 presents the number of classes that contain each type of Homogeneous Agglomeration. The first column presents the agglomeration type. The second column shows the absolute number of agglomerations found. The third column presents the participation of each Homogeneous Agglomeration in all agglomerations found, including Heterogeneous ones.

Table 5: Homogeneous Agglomerations on Qualita Corpus

Code Smell	Count	Participation in All Agg.
Feature Envy (FE)	528	34.7%
Long Method (LM)	25	1.6%
Total	553	36.3%

In total, we have found 1,523 code smell agglomerations, 553 of them being Homogeneous and 970 being Heterogeneous. We can observe from Table 5 that Homogeneous FE is indeed frequent, with almost 528 classes being affected by them, and its participation in all agglomerations is almost 35%. On the other hand, Homogeneous LM is very rare, representing only 1.6% of all agglomerations found.

4.3 Exploring Heterogeneous Agglomerations

Table 6 presents the itemsets found in the Qualita Corpus dataset. The first column presents the itemset with the smells that composes it. In the second column, we present the obtained Support. Itemsets consider all possible agglomerations that have certain smells, and consequently, those agglomerations can be a subset of larger agglomerations. The third column presents the Confidence, while forth column shows the supersets that support the presence of the itemset found by the algorithm and their respective percentage of participation in all Heterogeneous Agglomerations.

We highlight that we make a distinction between itemsets and Heterogeneous Agglomerations. The first presents a + sign. The agglomerations are represented by text. The main difference between the two is that itemsets are subsets, indicating that they are supported by different agglomerations. For instance, FE+LM is an itemset that was discovered by the algorithm, and is formed by all agglomerations that have FE and LM, *i.e.*, the FeLm, LcFeLm, LcRbFeLm, and RbFeLm agglomerations. Meanwhile, FeLm is the agglomeration formed by only FE and LM.

For example, the itemset LC+LM has a Support of 16.9%, and four agglomerations types support it: LcRbLm, LcRbFeLm, LcLm, and LcFeLm, with the last one being the most frequent (89 occurrences). However, when observing Confidence, we can conclude that we can not be confident about this result, that is, they may represent noise or minority cases [36]. Even though this itemset is not meaningful, for comparison purposes, we present its measurements.

We can observe that most meaningful itemset found are composed of two smells, one smell at the class level and another at the method level. They are often combinations with the FE smell. This may be due to the high presence of this smell in our dataset. The most frequent one is the LM+FE itemset, followed by LC+FE, with Support of 0.472 and 0.307, respectively. We also observe that for most itemsets, except LC+LM, the agglomeration that mostly contributes to the Support is the same agglomeration presented in the first column. For the LC+LM pair, the most frequent agglomeration is LcFeLm. This may be due to the high and unbalanced presence of the FE smell in this dataset, making the LcFeLm more frequent. To increase validity, we have changed the threshold values (Support) of the algorithm to verify if it gives different results, and even after changing the parameters, we did not find different

Table 6: Heterogeneous Agglomerations Found on Qualita Corpus

Itemset	Support	Confidence	Agglomeration	%Participation
LM+FE	0.472	0.782	LcRbFeLm	15 (0.015)
			RbFeLm	48 (0.049)
			LcFeLm	89 (0.092)
			FeLm	*306 (0.315)*
LC+FE	0.307	0.788	LcRbFeLm	15(0.015)
			LcRbFe	31 (0.032)
			LcFeLm	89 (0.092)
			LcFe	*163 (0.168)*
RB+FE	0.271	0.733	LcRbFeLm	15 (0.015)
			LcRbFe	31 (0.032)
			RbFeLm	48 (0.049)
			RbFe	*170 (0.175)*
LC+LM	0.169	0.434	LcRbLm	8 (0.008)
			LcRbFeLm	15 (0.015)
			LcLm	52 (0.054)
			LcFeLm	*89 (0.092)*

itemsets. Most itemsets found in this work were different from those found by Walter et al. [22], and LM+FE was also found by Palomba [25].

4.4 Impact of Agglomerations on Modularity

This section investigates the impact of agglomerations on the code modularity. We have selected six metrics that measure aspects of modularity: McCabe and maxNest measure the complexity, CBO the coupling of a class, LCOM* the cohesion, and finally, NOC and DIT characterize the inheritance use. The results are presented in Figure 2. Each square represents the information of a pair of agglomerations in row i and column j. The blue color shows cases when we rejected the null hypothesis for the metric, *i.e.*, the central tendencies of the categories are not equal, consequently, they impact the metric differently. Meanwhile, the red color represents that we could not reject the null hypothesis. The different shades of blue show the effect sizes. Notice that the red color has only one shade, since we could not reject the null hypothesis, we cannot present further information about the effect size. Meanwhile, for the blue color, we have four shades: Negligible ($|\text{delta}| < 0.147$); Small ($0.147 \leq |\text{delta}| < 0.330$); Medium ($0.330 \leq |\text{delta}| < 0.474$); and Large ($0.474 \leq |\text{delta}|$) [64]. The blank squares represent that we are comparing the same category.

In the heatmaps, we also present for each pair that rejected the null hypothesis if (i) Cliff's delta was positive ("+" sign) towards the category in the row, indicating that the row category usually has higher metrics values than the column; and (ii) if the delta was negative ("-" sign), it indicates that the category on the row usually has lower metric values than the category on the column. For example, for the LCOM* metric (Figure 2.(d)) we have that the pair Heterogeneous (on the row) and Homogeneous LM (on the column) has a "+" sign, indicating that Heterogeneous Agglomerations

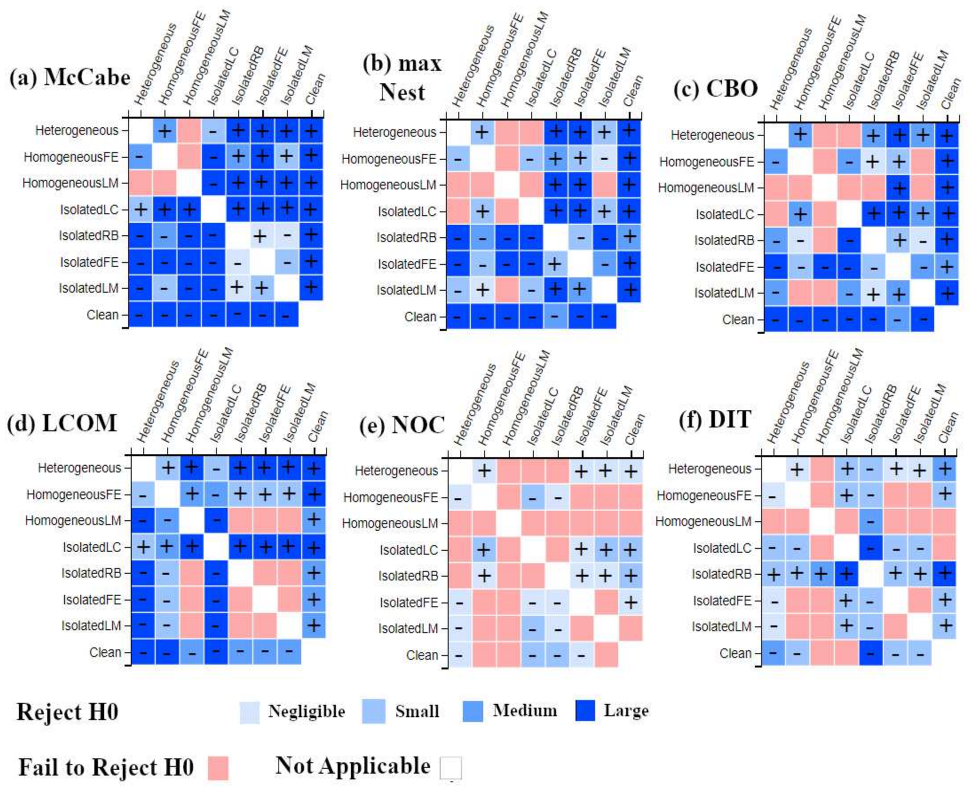

Fig. 2: Qualita Corpus Heatmaps for Effects of Agglomerations

has higher LCOM than Homogeneous LM, i.e. Heterogeneous Agglomerations are less cohesive than Homogeneous LM.If we check Homogeneous LM on the row and Heterogeneous Agglomeration on the column, we have a "-" sign, indicating the opposite. Figure 2 has six heatmaps, one for each of the analyzed metrics.

We can observe from the six heatmaps that for most metrics and pairs of agglomerations, we could reject the null hypothesis, except for the NOC metric (Figure 2.(e)). That is, there is a significant difference between the metrics values for different agglomeration types. We can highlight that for the NOC and DIT metrics, most of their effects are small, indicating that the difference between their behavior is more uniform than other metrics. This may be explained by the values for these metrics being usually low. However, we can observe that for the DIT metric, the Isolated RB has all of its effects as positive.

For most modularity aspects, except inheritance, we have found that when a class has at least one code smell, its impact is statistically higher than when the class is Clean (all squares of Clean classes are with a "-" sign). Even more, in general, classes with at least two smells (check the "+" signs for the Heterogeneous, Homogeneous FE) present a higher impact on the complexity and coupling (McCabe+maxNest, and

CBO, respectively). For the Homogeneous LM, we can observe that the coupling and complexity tend to be higher when compared to Isolated FE and Clean Classes. For cohesion, we can also check that Heterogeneous and Homogeneous FE are less cohesive than other categories.

It is interesting to notice that Isolated LC is the only isolated smell that impacts more than some agglomerations, since most of the squares have a "+" sign, even when compared against Homogeneous FE and Homogeneous LM. This provides us initial evidence that even when a class has multiple FE and LM, the class usually has lower complexity and coupling when compared to a class that has only an LC. This finding seems contradictory, since the FE smell is a method that is highly coupled to another class, and LM has a high number of lines of code. This result may be due to the other methods in the class that lower the metric value. For instance, for the maxNest metric (Figure 2.(b)), we could not reject the null hypothesis between Isolated LC and Homogeneous LM. This may be because maxNest of a class is the highest nesting of conditionals/loops in all class's methods. Consequently, since the algorithm identify the method that had the highest nesting, clean methods does not affect this metric, since they usually have low nesting values.

For the inheritance aspect, we can observe that Isolated RB and Heterogeneous, in general, had a higher impact on this aspect (high presence of "+" signs). This is interesting because we expected that when the hierarchy is too deep, the more the class would reject their parents' behavior, consequently presenting RB. We also observe that the impact of Isolated LC on this aspect was mostly negative for the DIT, but they tend to have more children than other categories (Figure 2.(e)).

Finding 1: For the Qualita Corpus dataset, we could identify that Heterogeneous, Homogeneous Agglomerations, and Isolated LC indeed impact the most on class modularity. However, when isolated, the LC smell has a higher impact on complexity, cohesion and coupling when compared to Homogeneous FE and Homogeneous LM.

4.5 Impact of Heterogeneous Agglomerations on Modularity

This section investigates which Heterogeneous Agglomeration mostly affects modularity, and those that have a uniform behavior in the Qualita Corpus dataset. Consequently, we can provide initial evidence of which agglomeration should be prioritized in the refactoring process. Figure 3 presents six heatmaps with the Cliff's Delta effects for each category of Heterogeneous Agglomeration found in the Qualita Corpus. Each heatmap concerns one of the evaluated metrics. For legibility purposes, the agglomerations are categorized according to the smells that they contain. For instance, *LcFe* is an agglomeration composed of only LC and FE.

We can observe in Figure 3 that for most heatmaps, we have a high presence of red squares, indicating that we could not reject the null hypothesis, except for McCabe metric (Figure 3.(a)). This means that when compared against each other, the computed metric for the categories did not present a statistically significant difference in their distribution. However, for all metrics, we have identified that the different Heterogeneous Agglomerations have a similar behaviour. However, agglomerations that

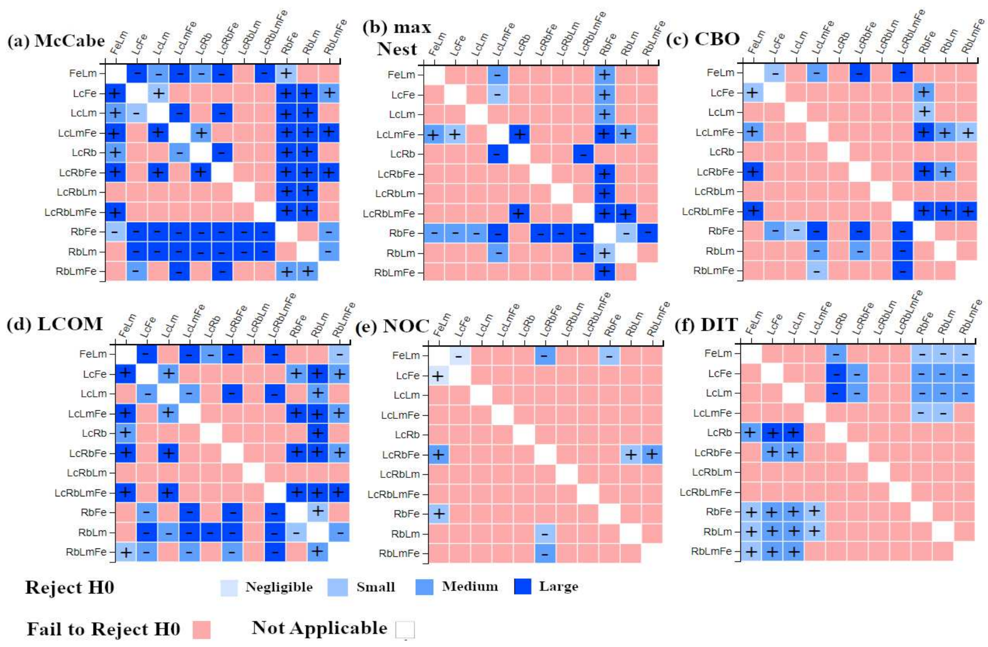

Fig. 3: Qualita Corpus heatmap for Effects of Heterogeneous Agglomerations

have an LC smell tend to have a higher impact on complexity and cohesion than those that do not have it. We also observe that for complexity (Figure 3.(a)) and cohesion (Figure 3.(d), usually the FeLm category is less complex and more cohesive. We can also observe that RbFe concentrates a large number of the "-" signs, mainly when compared to agglomerations that present an LC smell. However, for the DIT measure, agglomerations that do not have LC are usually higher on the inheritance hierarchy when compared to RbFe and RbLm (Figure 3.(f)).

Finding 2: We have not found a significant difference in impact for the different categories of Heterogeneous for all analysed metrics in the Qualita Corpus dataset. However, agglomerations that have LC smell tends to have a higher complexity and lower cohesion than those that do not have it.

5 GitHub Results

5.1 Overview of Smells Found

Table 7 is organized in the same way as Table 4. We can observe that the most frequent smell on this dataset is *FE*, with more than 40% of participation on the total number of smells, followed by the *LM* smell, with almost 32% of participation. Consequently, 72% of the smells are at the method level (i.e. FE and LM), and 28% at the class level (i.e. LC and RB). However, when we consider all classes/methods, the smells are indeed rare: LC and RB affect almost 2% of the classes, and FE and LM affect less than 1% of the total methods.

Table 7: Code Smell Found

Code Smell	**#Smell**	**#Smell/#TotalSmell**	**% Smelly Elements**
Large Class (LC)	194	14.27%	1.85%
Refused Bequest (RB)	189	13.62%	1.77%
Feature Envy (FE)	564	40.63%	0.67%
Long Method (LM)	441	31.77%	0.52%
Total	1,388	100%	-

5.2 Exploring Homogeneous Agglomerations

Table 8 is organized in the same way as Table 5. In total, we have found 266 code smell agglomerations, with 90 being Homogeneous and 176 Heterogeneous. We can observe from Table 8 that all Homogeneous Agglomerations were indeed frequent in the source code, with FE being the most frequent one, with about 23% of all the agglomerations being of this type. We notice that the presence of Homogeneous LM is also significant, with the participation of 10% considering all agglomerations. In total, Homogeneous Agglomerations represent almost 34% of all agglomerations found for this dataset.

Table 8: Homogeneous Agglomerations on GitHub

Code Smell	Count	Participation in All Agg.
Feature Envy (FE)	62	23.3%
Long Method (LM)	28	10.5%
Total	90	33.8%

5.3 Exploring Heterogeneous Agglomerations

Table 9 presents the Heterogeneous Agglomerations, organized in the same way as Table 6. We observe in Table 9 that four itemsets were more meaningful, all of them being composed of two smells, and with at least one smell at the method level. The most frequent itemset was LC+FE, with Support of 0.352 and Confidence of 0.585. For this itemset, we had three types of agglomerations: LcRbFeLm, LcFeLm and LcFe, with LcFe being the most frequent, with the participation of 0.244 out of the Support of LC+FE (0.352). We highlight that for all itemsets, the agglomeration that mostly contributes to its Support is of the same type as the itemset. For instance, for itemset LC+FE, the highest participation is the agglomeration LcFe (43%). Observing the Confidence measure, all itemsets found were meaningful (above 0.5), with the highest Confidence for the RB+FE itemset (0.674).

Table 9: Heterogeneous Agglomerations Found on GitHub

Itemset	Support	Confidence	Agglomeration	%Participation
LC+FE	0.352	0.585	LcRbFeLm	2 (0.011)
			LcFeLm	17 (0.097)
			LcFe	*43 (0.244)*
LM+FE	0.347	0.560	LcRbFeLm	2(0.011)
			RbFeLm	7 (0.040)
			LcFeLm	17 (0.097)
			FeLm	*35 (0.199)*
LC+LM	0.335	0.557	LcRbLm	2 (0.011)
			LcRbFeLm	2 (0.011)
			LcFeLm	17 (0.097)
			LcLm	*38 (0.216)*
RB+FE	0.165	0.674	LcRbFeLm	2 (0.011)
			RbFeLm	7 (0.040)
			RbFe	*20 (0.114)*

5.4 Impact of Agglomerations on Modularity

This section explores how agglomerations compare between themselves and with classes that are clean or have only one smell. This comparison is made through the evaluation of the selected modularity metrics (see Section 3.7 – Table 3). Figure 4 presents six heatmaps, one for each of the metrics. We can observe from Figure 4 that for most modularity aspects, except the inheritance (NOC and DIT metrics), we reject the null hypothesis (blue squares), and most of the effects were large (darkest blue). For all analyzed modularity aspects, we can observe similar results: (i) Clean classes tend to have negative effects ("-" signs), indicating that the presence of code smells in the code hinders the modularity; (ii) most effects for Heterogeneous Agglomerations are positive ("+" signs), indicating that they may be more harmful to code modularity, and may make maintenance harder and more costly; and (iii), when isolated, the LC smell presents a higher impact, with Isolated LC having most of its blue squares with a "+" sign. In fact, the only negative effects with the Isolated LC found were when compared to the Isolated RB for the DIT metric.

Interestingly, we have also found that Homogeneous LM impacts more than Isolated smells, except when compared to Isolated LC, on the complexity and coupling. Homogeneous FE had a higher impact only when compared to Clean classes. We can also observe that for the inheritance (Figure 4.(e) and 4.(f)) we could not reject the null hypothesis for the pairs with Clean classes. For instance, the pairs Clean-Homogeneous FE and Clean-Homogeneous LM This is interesting and reflects the smell's nature, since FE and LM are not associated with inheritance. For all pairs with Isolated RB, we have found a positive effect towards it (see the "+" signs), indicating that usually, their impact on inheritance is higher than other categories, with mostly large effects.

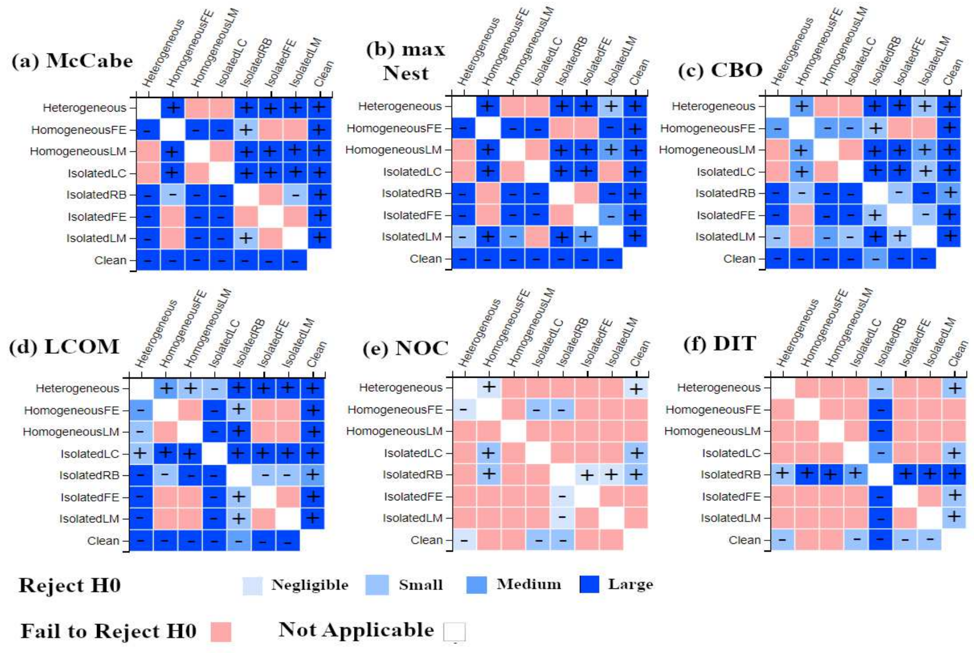

Fig. 4: GitHub Heatmap for Effects of Agglomerations

Finding 3: For the GitHub dataset, we can conclude that Heterogeneous Agglomerations and Isolated LC impact the most on modularity in the GitHub systems. Consequently, they tend to be more complex and more coupled to other classes and may make evolution and maintenance activities costlier.

5.5 Impact of Heterogeneous Agglomerations on Modularity

In this section, we investigate if there is an agglomeration that contributes the most to the impact of Heterogeneous Agglomerations. Figure 5 presents six heatmaps, one for each of the analyzed metrics. These heatmaps can be interpreted the same way as Figure 4. Notice that we could not find any LcRbFe agglomeration in this dataset. From Figure 5, we can see evidence that the difference between Heterogeneous Agglomerations is not conclusive, with most squares being red (not rejecting null hypothesis). This may be due to the fact that some of the categories did not have the necessary instances to compute the statistical tests, even with corrections being applied. For the blue squares, we observe some large effects, mostly in combinations with LcFe, LcLm, LcLmFe, RbFe, and RbLm. However, with respect to their effect size, they are mostly negative for RbFe and RbLm, except for the RbFe in the DIT metric, achieving positive effects when compared to agglomerations that do not present RB smell.

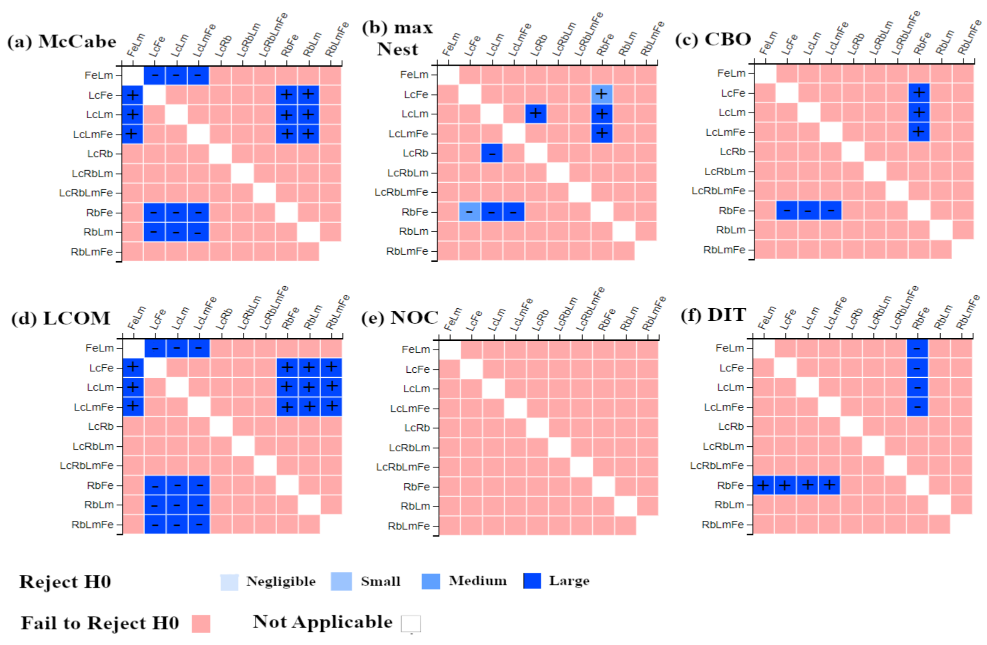

Fig. 5: GitHub Heatmap for Effects of Agglomerations

Finding 4: For the GitHub dataset, we could not find a significant difference between the Heterogeneous Agglomerations.

6 Comparison Between Datasets

6.1 Homogeneous Agglomeration Comparison

In Sections 4.2 and 5.2, we have found that Homogeneous Agglomerations are indeed frequent on both datasets, mainly the FE one, with a participation of almost 35% on Qualita Corpus, and 23% on GitHub. We have also observed that Homogeneous LM has a significant presence on GitHub (about 10%). However, the high presence of Homogeneous Agglomerations is not expected, since their appearance may indicate the presence of other smells. For instance, a class that has two or more LM may indicate the presence of an LC smell.

Following this rationale, we discuss the presence of Homogeneous FE, since this agglomeration has significant participation in both datasets. One of the causes for their presence may be the detection strategies used by the detection tools. However, we first discuss the number of smells present in these agglomerations.

Figure 6 presents, for both datasets, the distribution of Homogeneous FE on the systems, and how many smells usually compose it. Boxplots with the prefix “QC” indicate data related to the Qualita Corpus dataset. Boxplots with the prefix “GH” indicate data related to the GitHub dataset. Item (a) presents the boxplots related to the number of Homogeneous FE in each system of the respective dataset. Item (b) presents the boxplots related to the number of FE smells on each Homogeneous FE Agglomeration at the respective dataset.

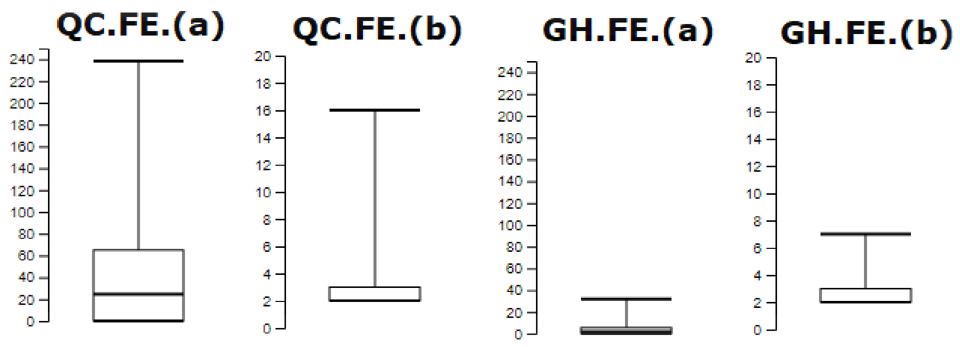

Fig. 6: Presence of Feature Envy smell on the Homogeneous FE Agglomerations. (a) Number of FE Agglomerations in each system of the dataset. (b) Number of FE smells on each Homogeneous FE Agglomeration of the dataset.

It is expected that larger systems have more "opportunities" of presenting agglomerations than smaller ones. Observing Figure 6.QC.FE.(a), the number of classes affected by such agglomerations varies greatly, with most systems having 0 to 60 agglomerations, but the median is of about 20 instances per system. The system that had the most agglomerations was Weka, with 232 Homogeneous FE Agglomerations.

Meanwhile, for the GitHub dataset, we can observe in Figure 6.GH.FE.(a) that most systems have 2 to 5 Homogeneous FE agglomerations, and the system that had the most agglomerations was libGDX, with 32 Homogeneous FE. However, the median was about 3 agglomerations per system. Overall, when observing the number of FE smells that compose such agglomerations, we can see that for most smelly instances the value is low, in both datasets (Boxplots 6.QCFE.(b) and 6.GH.FE.(b)) with most agglomerations being composed of two to three FE smells. These low values may be one of the reasons that such agglomerations do not present other smells.

Finding 5: (RQ1. Which Homogeneous Agglomerations are more frequent in the source code?) For both datasets, we have found that they are indeed frequent, mainly the Homogeneous FE agglomeration, with more than 20% of participation in both datasets. Even though Homogeneous LM did not have an expressive presence on Qualita Corpus, it did on the GitHub dataset. Consequently, since Homogeneous Agglomerations are not rare, they should be further investigated if method smells are evaluated at class level.

6.2 Heterogeneous Agglomerations Comparison

Sections 4.3 and 5.3 presented Heterogeneous Agglomerations found using Itemset Mining Algorithm. On both datasets, we have found four Heterogeneous Agglomerations with a significant presence on the source code: LC+LM, RB+FE, LC+FE, and FE+LM. However, the order in which they contributed was different, with FE+LM being the most present on Qualita Corpus (Support of 0.472), and LC+FE being the most present on GitHub (Support of 0.352). However, when observing the Confidence, LC+LM had its value below 0.5, indicating that for the Qualita Corpus dataset, this itemset was not meaningful, while for GitHub its Confidence is 0.557.

We can also observe that for all these itemsets, except LC+LM in Qualita Corpus, the agglomeration that contributed the most for the high Support was the same agglomeration on the itemset. For instance, Table 9 presents Heterogeneous Agglomerations for the GitHub dataset showing that the LcLm agglomeration contributed the most for the LC+LM itemset, with a participation of 0.216 out of 0.335. For the LC+LM itemset on Qualita Corpus (Table 6), the agglomeration LcFeLm contributed the most for LC+LM Support, with a participation of 0.092 out of 0.169.

Finding 6: (RQ2. Which Heterogeneous Agglomerations are more frequent in the source code?) For both datasets, we have found that all the heterogeneous itemsets were composed of two smells. They were: LC+LM, RB+FE, LC+FE, and FE+LM, but appeared in different orders due to differences in their Support. The most frequent ones were LC+FE for GitHub, and FE+LM for Qualita Corpus. However, when considering their Confidence, for the Qualita Corpus, the itemset LC+LM had its Confidence below 0.5, indicating that it may not be as meaningful as all the other itemsets.

6.3 Comparing the Code Smell Agglomeration's Impact

In this section, we compare the impact of agglomerations, highlighting their main differences and similarities, according to their results on the rejection of the null hypothesis. Our null hypothesis stated that there is no difference between the distribution of the metric values for a pair of categories.

For this purpose, Figure 7 presents six heatmaps, one for each of the analyzed metrics, presenting the differences and similarities of the results obtained for both Qualita Corpus and GitHub datasets. White squares indicate that we could not compute the tests, *i.e.*, we are comparing the same agglomeration type. The lightest gray squares indicate that we could reject the null hypothesis in one dataset, but not on the other, providing us partial evidence about the agglomeration's modularity impact. Gray squares with a "∼" symbol indicate we reject the null hypothesis in both datasets, but they had disagreed with the effect size classification. The gray cells with the "=" symbol indicate that we reject the null hypothesis in both datasets, and they agreed on the effect size classification. Finally, black squares with an "X" symbol indicate that we did not reject the null hypothesis in either dataset.

We can observe in Figure 7 that in almost all heatmaps, except NOC and DIT (Figure 7.(e) and (f)), we confirm the results in both datasets through their agreement

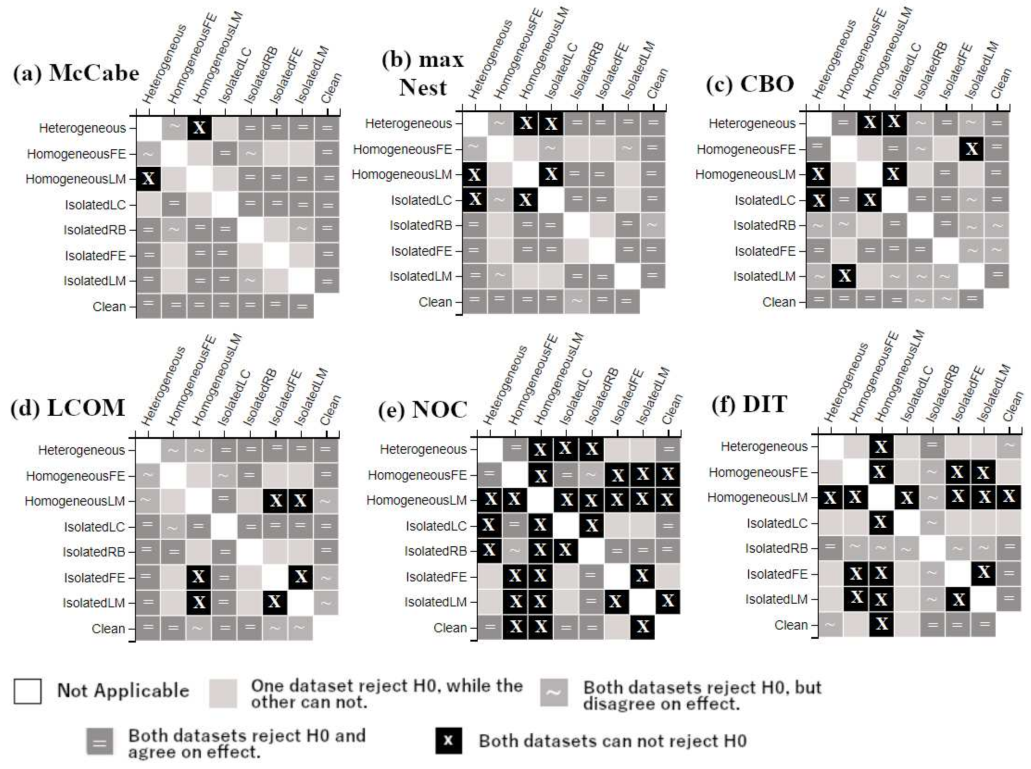

Fig. 7: Differences Between the Two Dataset's Metric Impact

on the statistical test ("∼" and "=" squares). For the DIT heatmap (Figure 7.(f)), we mostly have partial evidence of the inheritance impact (light gray squares). However, for Isolated RB, we could confirm that the presence of RB has an impact on the inheritance aspect for both NOC and DIT metrics, mainly when compared to Isolated and Clean classes.

For the McCabe, maxNest, CBO, and LCOM metrics (Figure 7.(a), (b), (c) and (d), respectively), most of the squares were"=", indicating that we could confirm the impact of the agglomerations in both datasets, and they agree in both statistical test and effect size. In general, we can observe from all heatmaps that most of the agreement is on rejecting the null hypothesis for both datasets (dominance of "∼" and "=" signals), and they most agree on the effect size for Heterogeneous Agglomerations, Clean Classes, Homogeneous LM (McCabe and maxNest), and the Isolated LC (McCabe, maxNest, CBO and LCOM).

Finding 7: (RQ3. How do code smell agglomerations impact the system modularity?) For most metrics, with inheritance being an exception (NOC and DIT metric), we could observe that both datasets agree with rejecting the null hypothesis (see “∼” and “=” signals). Moreover, they agree on the effect size (“=” signal), *i.e.,* we could confirm that different datasets lead to similar results. Moreover, we could also observe that most of these agreements are in relation to Heterogeneous Agglomerations and Clean classes, with Heterogeneous usually having large and positive effects. Consequently, we have found evidence that (i) agglomerations do impact the code modularity, mainly on complexity, cohesion, and coupling aspects; and (ii) most of the results are confirmed for both datasets.

6.4 Comparing the Heterogeneous Agglomeration’s Impact

This section discusses the results found for the Heterogeneous Agglomerations’ impact on modularity. Our main goal with this question is to understand if different combinations of code smell present in an agglomeration have different impacts on the analyzed metrics.

For the Qualita Corpus dataset, we could mostly see differences in the McCabe and LCOM metrics. We could also observe that agglomerations that have LC smell tend to have a higher metric value than agglomerations that do not have it. For instance, RbFe and RbLm had all of their effects negative in the McCabe and LCOM metric when compared to classes that had LC (Figure 3).

For the GitHub dataset, for all analyzed metrics, we could not reject most of the null hypotheses. The pairs that rejected the null hypothesis and had a large effect were LcLm, LcFe, and LcLmFe (Figure 5). We also noticed that the LcRbFe agglomeration did not appear on the GitHub dataset. Meanwhile, for the Qualita Corpus dataset, we can observe that McCabe and LCOM metrics have more rejection of the null hypothesis, with most of its large effects when compared to the RbFe and RbLm agglomerations.

Finding 8: (RQ4. Do the different types of Heterogeneous Agglomerations have a uniform impact on the system modularity?) For both datasets, we have not found statistical differences between the different agglomerations for the CBO, DIT, NOC, and maxNest metrics. However, for the McCabe and LCOM metrics on the Qualita Corpus dataset, we could observe a higher presence of large effects. Although we did not find a significant difference in the metric behavior of Heterogeneous Agglomerations, we provide partial evidence that some agglomerations behave differently on the Qualita Corpus dataset. This may be due to the fact that the versions of the systems of the dataset are older than the GitHub ones.Consequently, the available technologies and support tools may influence the proportion of smells found. We also highlight that the domain, maturity, and developing practices also affect the presence of code smells, and consequently, the presence of agglomerations. 6.5

6.5 Discussion

This section discusses our findings and their implication for practitioners. First, results indicate that Heterogeneous Agglomerations affect more the modularity than Isolated Smells and Homogeneous Agglomerations, except the Isolated LC. Consequently, they affect the system comprehension and increase the cost and maintenance effort. This implies that developers need to pay attention to this agglomeration type when doing their maintenance activities. Consequently, automatic detection tools should consider the presence of such agglomerations in the source code in order to better support development activities. We also observe from our analysis that Homogeneous Agglomerations have a significant presence on the source code, and they indeed impact software modularity. This evidence implies that developers need to pay attention to how code smells interact (forming an agglomeration) when modifying the code, since these relations may be complex.

Another analysis worth discussing is the varying number of smells and agglomerations present in both datasets. Figure 8 shows how the smells are distributed on the systems in order to mitigate some of the bias created by the difference in the size of the datasets. Figure 8 presents four sets of boxplots, one set for each of the analyzed smells. For each set, we present two boxplots representing the percentage of classes/methods affected by the smell on each system on the Qualita Corpus (QC) and GitHub (GH) dataset.

For the FE smell (FE.QC and FE.GH), we can observe that the difference in the frequency is very high, with a median of 3% for Qualita Corpus (FE.QC) and about 0.7% for GitHub (FE.GH). This was the highest difference in the median frequency for all analyzed smells, followed by RB, which for Qualita Corpus was about 3.5% (RB.QC), and for GitHub, it was 0.75% (RB.GH). For LM and LC, we obtained a similar median value. For the LM, the frequency, in general, was about 0.4% for both datasets (LM.QC and LM.GH). Meanwhile, for LC, the difference in the median values was less than 1% (LC.QC and LC.GH).

Consequently, for two out of four smells, we have a similar distribution (LM and LC), while for FE and RB their frequency distribution was different, and they may have directly influenced the results found. One reason for this result may be the system domain and size. Tables 1 and 2 reflect these aspects. In the Qualita Corpus (Table 1), most systems had more than 20KLOC, while in the GitHub (Table 2) most systems had 5KLOC to 10KLOC. However, since we have found that most of the results hold for both datasets, except the impact of Heterogeneous Agglomerations, we believe these factors did not represent a critical bias to our results. Our intention for the creation of the GitHub dataset was to vary sizes, domains, and levels of maturity, beyond verifying if the newest techniques used in open-source development influence our results.

Other factors that may be affecting our results are the different levels of system maturity and the practices adopted in the open-source community. For instance, the systems of the Qualita Corpus dataset were obtained in 2013, while the GitHub ones have been currently maintained and evolved. This difference in the time span ranges different open source development techniques and quality criteria, such as refactorings, code styles, tool support, code review, gate keeping and testing [65, 66]. For example,

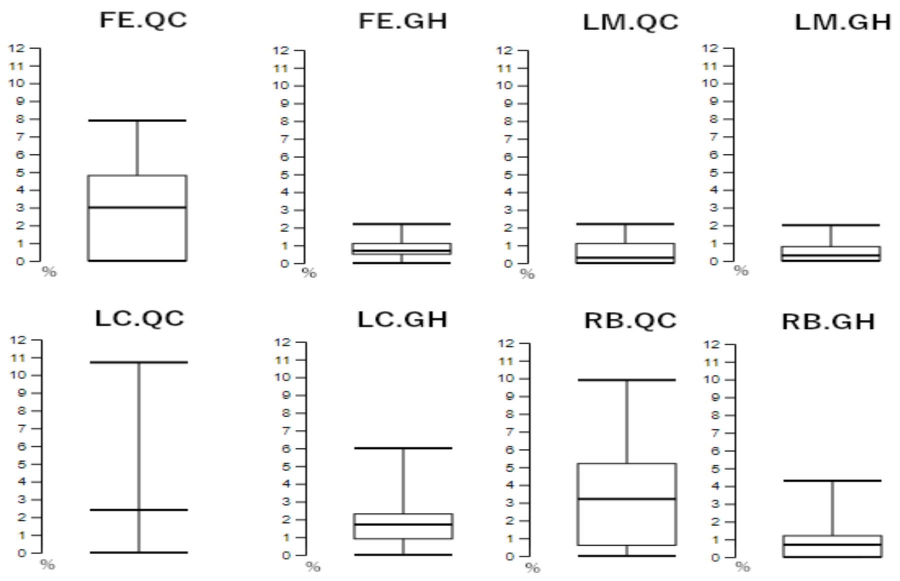

Fig. 8: Participation of Each Smell on the Selected Systems For Each Dataset

GitHub allows developers and bots to recommend changes and refactorings to pull requests, increasing the code quality and code style guidelines [16, 38, 67, 68, 69, 70]. Consequently, these different ecosystems practices may directly impact the presence of code smells. This seems the case for the Heterogeneous in-depth analysis of the GitHub dataset.

7 Threats to Validity

This section presents the threats to the validity of our work, organized as suggested by Wohlin [71].

Internal Validity. One possible threat to the internal validity is the metrics used in our analysis. To mitigate this threat, we used hypothesis testing. We used Mann-Whitney U to test where the distribution of the metrics was equal for the different types of agglomerations. We opted for using the Mann-Whitney U test since our data does not follow a normal distribution. In cases where we have less than eight instances, mainly on the heterogeneous analysis, the algorithm is parametrized to make the necessary corrections. We also combine the test with Bonferroni Correction, given that we are comparing more than two categories at the same time. Finally, we calculate the Cliff's Delta, a measurement of effect size to provide us information on which distribution has higher metrics values.

We selected four smells for our analysis, which may also be a threat to the internal validity of our study. We have evaluated two smells at the class level and two smells at the method level. We selected them considering previous works that confirmed their high impact [9, 12, 13, 14, 15, 72, 26]. Thus, we believe that we have covered relevant smells for practitioners. However, we are aware that the obtained results for their impact on modularity may not hold for agglomerations formed by other types of smells. Notice that the same is valid for the selection of the modularity metrics. We also selected the used metrics, influenced by their use in the literature [21, 59] and evidence of its correlation with the modularity aspects [13, 60].

Construction Validity. To classify a smelly/non-smelly code fragment, we relied on a set of automatic detection tools widely used in the literature [21] and composed of different detection strategies. However, we are aware that some code fragments may be misclassified as smelly/non-smelly. To mitigate this threat: (i) we previously manually validated a sample of code smells with developers in order to understand how accurate our strategy is [33]; (ii) we selected the tools according to the strategy they used for the smell identification, avoiding using tools that used in their detection strategy the analyzed metrics; (iii) we consider as smelly if at least two out of three tools detect the instance as positive. Thus, the bias created by the tools is mitigated in the voting system (see Section 3.5). We then believe we mitigate any bias introduced in the construction of the dataset.

External Validity. The results of the paper depend directly on how the dataset was constructed. To achieve a more generalizable result, we opted to create a new dataset that contains systems from different sizes and domains. In both phases of our study, we have extracted information about 30 open source Java systems, 20 from Qualita Corpus and 10 from GitHub. However, we cannot claim the results generalize for other systems and languages. We selected a different number of systems from two datasets, Qualita Corpus and GitHub, which might be a threat. The Qualita Corpus has two times more classes and methods than GitHub. Moreover, the systems in the Qualita Corpus are older than the GitHub ones, presenting different levels of maturity, sizes, and domains. However, we aimed to explore whether the different practices in both current open-source development datasets affect the agglomerations. We also highlight that using a different set of metrics may lead to different results. However, we present initial evidence that agglomerations in general are more harmful to modularity than classes with only one smell. Thus, they should be explored in depth in different settings. Even though the generalization of the study findings is limited to the smells and metrics we selected, we provided a detailed research methodology, allowing the replication of the study steps to investigate other smells and metrics.

Conclusion Validity. To better analyze our results, we first discussed the individual results for each dataset, and later compared their agreement. We carefully analyzed our data. When analyzing the results found, we presented and focused on the percentage of participation of agglomerations according to the dataset. Moreover, in the discussion section, we evaluated how the smells found are distributed by the system in each dataset. Since we are dealing with a large volume of data, we used consolidated algorithms from the R and Python languages. To avoid errors in transcription, all scripts and data were revised several times to ensure their correctness. Moreover, to ensure the replicability of our results, all scripts are available in our supplementary material [37].

8 Related Work

Several studies aimed to understand the relationship between code smells and their co-occurrence [18, 19, 22, 24, 25, 28, 31, 73]. In their book, Lanza and Marinescu [18] identified that code smells are related to each other, introducing the Collaboration Disharmonies concept, in which design flaws interact and affect several entities at once. To identify such relationships, the authors used metrics of coupling. In this work, our goal is to understand how agglomerations can impact software modularity.

Oizumi et al. [19] studied how code anomalies relationships help developers identify design problems, and found that some agglomerations are related to design problems. Palomba et al. [9] found that when a class has more than one smell, they are more prone to faults and changes. Hamdi et al. [74] identified code smell agglomerations on Android applications. The authors considered agglomerations formed by smells specific to Android apps and Object-oriented smells. We complement these works by presenting evidence of how agglomerations on Java projects may impact the source code modularity.

Oizumi et al. [31] studied the relationship between agglomerations and architectural problems in seven systems. The authors found that most of the architectural problems are related to agglomerations, concluding that they are better at indicating the presence of such problems. Fontana et al. [73] studied how code smell and architecture smells correlate. They have found strong correlations for only a few of the analyzed systems. Differently from these works, our focus is on identifying object-oriented code smell agglomerations, and we do not evaluate system architecture.

Martins et al. [32] studied how code smell agglomeration's refactoring impacted the metric values of three industrial systems. Beyond identifying which aspects of quality the refactoring improved, the authors also asked developers which agglomerations were the hardest to remove. Vidal et al. [75] proposed criteria to rank the agglomerations, considering their type and their modifiability. These criteria were evaluated, considering how these agglomerations help developers to identify architectural problems. In this work, we focus on understanding if the agglomerations in the same class have different behavior in terms of modularity metrics.

Lozano et al. [28] studied the life-cycle of code smell agglomerations, *i.e.*, when code smells co-exist and co-disappear. Palomba et al. [25] investigated code smell co-occurrence using association rules in 30 systems. Later, Palomba et al. [27] investigated several system versions to understand how agglomerations behave in the source code. The authors found that most smelly classes contain agglomerations, but they tend to disappear together when removed from the code.

Yamashita and Moonen [30] investigated the impact of code smell agglomerations on the source code maintainability using Principal Component Analysis (PCA). The authors identified 15 smells using one detection tool. They found that certain types of agglomerations impact the maintenance activity. Even though we use a different technique, we complement this work by providing evidence that agglomerations also impact software modularity.

Walter et al. [22] evaluated code smell agglomerations on the Qualita Corpus [34], raising evidences that the system domain affects the result found. They also presented a comparison between the new agglomerations found by them and those that were already found in the literature. The authors used 9 tools to detect 14 smells. They separated their dataset to mitigate tool bias. However, their separation is not consistent for all smells due to the number of tools that detect them. We complement this work by presenting an analysis of how agglomerations impact modularity at two levels: a general one, and an in-depth analysis of the Heterogeneous Agglomerations. We also use a voting system to mitigate the tools bias, using 3 tools for each smell analyzed. Furthermore, we include a novel type of agglomeration in our analysis: the homogeneous ones.

In this paper, we extend the knowledge from previous work [18, 19, 22, 24, 25, 28, 31]. Differently from other authors, we focus not only on identifying the agglomerations but also on their modularity impact. In other words, we aim to understand how agglomerations can impact different aspects of modularity. We also further classified them considering their different types, *i.e.*, Heterogeneous and Homogeneous Agglomerations.

9 Conclusions and Future Works

This work aimed at evaluating how code smell agglomerations impact the source code modularity. For this purpose, we compare the results found for the 20 systems extracted on Qualita Corpus [34] from previous study [33], with an additional mining of 10 popular systems from GitHub. This analysis allowed us to understand if the results hold for different datasets, and how the technology gap influences the results found. We then collected the modularity metrics and the code smells evaluated (LC, RB, LM, and FE). In order to characterize if agglomerations are indeed frequent on the systems, we have used frequent itemset mining to detect the Heterogeneous ones, and frequency statistics to identify the Homogeneous ones. To calculate the impact, we used a hypothesis test, which provides us with information about the differences between the behavior of the metrics for different agglomeration types. These analyses allowed us to identify which agglomeration impacts the different aspects of the modularity of the systems.

Our main results and contributions are: (i) we have found that Homogeneous Agglomerations have a significant presence in both datasets, but mainly the Homogeneous FE one; (ii) for both datasets, the itemsets LC+FE and FE+LM were the most frequent agglomerations in the code; (iii) Heterogeneous Agglomerations impact modularity the most, except on the inheritance aspect; (iv) both Homogeneous Agglomerations under study impacts the complexity and coupling when compared to Isolated Classes; (v) we did not find a statistical difference between the different types of Heterogeneous for both datasets.

This work provides initial evidence of how agglomerations impact the source code modularity. This information can be used to prioritize the refactoring of the most harmful agglomerations. In future work, we plan to expand our dataset to include systems from other domains and replicate the ensemble method to evaluate other smells. We also plan to expand our analysis to provide evidence that agglomerations impact fault and change proneness. We also highlight the necessity of investigating code smell agglomerations at: (i) industrial settings, since industrial systems may use different development processes, and these practices may affect directly the presence of agglomerations; (ii) other programming languages rather than Java, because different languages have different specificity, and consequently, they have different types of smells than Java ones. Consequently, their impact should be further evaluated.

Declarations

Funding. This research was partially supported by Brazilian funding agencies: CAPES, PROEX, 88887.595966/2020-00.

Declaration of Interest. We declare that we have not any conflict of interest when submitting this manuscript to Software Quality Journal.

Supplementary Data. All of the available data will be published under GitHub and are currently available at [37].

Consent. All the authors in this manuscript consent to its publication, and have read the journal guidelines.

Contributions. Amanda Santana: Study Design, Data Analysis, Manuscript Writing, Review. Juliana Alves Pereira: Critical Review. Eduardo Figueiredo: Study Design, Critical Review. Alessandro Garcia: Critical Review.

References

[1] Fowler M. Refactoring: Improving the Design of Existing Code. Addison-Wesley; 1999.

[2] Xia X, Bao L, Lo D, Xing Z, Hassan AE, Li S. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. IEEE Transactions on Software Engineering. 2018 Oct;44(10):951–976.

[3] Yamashita A, Counsell S. Code smells as system-level indicators of maintainability: An empirical study. Journal of Systems and Software. 2013;86(10):2639–2653.

[4] Sjøberg DIK, Yamashita A, Anda BCD, Mockus A, Dybå T. Quantifying the Effect of Code Smells on Maintenance Effort. IEEE Transactions on Software Engineering. 2013 Aug;39(8):1144–1156.

[5] Tufano M, Palomba F, Bavota G, Oliveto R, Di Penta M, De Lucia A, et al. When and Why Your Code Starts to Smell Bad. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 1; 2015. p. 403–414.

[6] Kerievsky J. Refactoring to Patterns. Pearson Higher Education; 2004.

[7] Yamashita A, Moonen L. Do developers care about code smells? An exploratory survey. In: 2013 20th Working Conference on Reverse Engineering (WCRE); 2013. p. 242–251.

[8] Uchôa A, Barbosa C, Coutinho D, Oizumi W, Assunçao WK, Vergilio SR, et al. Predicting design impactful changes in modern code review: A large-scale empirical study. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). IEEE; 2021. p. 471–482.

[9] Palomba F, Bavota G, Di Penta M, Fasano F, Oliveto R, De Lucia A. On the Diffuseness and the Impact on Maintainability of Code Smells: A Large Scale Empirical Investigation. In: 2018 IEEE/ACM 40th International Conference on Software Engineering. (ICSE); 2018. p. 482–482.

[10] Rahman MM, Satter A, Joarder MMA, Sakib K. An Empirical Study on the Occurrences of Code Smells in Open Source and Industrial Projects. In: Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '22. Association for Computing Machinery; 2022. p. 289–294.

[11] Politowski C, Khomh F, Romano S, Scanniello G, Petrillo F, Guéhéneuc YG, et al. A large scale empirical study of the impact of Spaghetti Code and Blob anti-patterns on program comprehension. Information and Software Technology. 2020;122:106278.

[12] Olbrich SM, Cruzes DS, Sjøberg DIK. Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems. In: 2010 IEEE International Conference on Software Maintenance; 2010. p. 1–10.

[13] Basili VR, Briand LC, Melo WL. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering. 1996 Oct;22(10):751–761.

[14] Khomh F, Di Penta M, Guéhéneuc Y, Antoniol G. An Exploratory Study of the Impact of Antipatterns on Class Change- and Fault-Proneness. Empirical Software Engineering. 2012 Jun;17(3):243–275.

[15] Hall T, Zhang M, Bowes D, Sun Y. Some Code Smells Have a Significant but Small Effect on Faults. ACM Transactions Software Engineering Methodology. 2014 Sep;23(4).

[16] Falessi D, Kazman R. Worst Smells and Their Worst Reasons. In: 2021 IEEE/ACM International Conference on Technical Debt (TechDebt); 2021. p. 45–54.

[17] Abbes M, Khomh F, Guéhéneuc Y, Antoniol G. An Empirical Study of the Impact of Two Antipatterns, Blob and Spaghetti Code, on Program Comprehension. In: 2011 15th European Conference on Software Maintenance and Reengineering; 2011. p. 181–190.

[18] Lanza M, Marinescu R, Ducasse S. Object-Oriented Metrics in Practice. Springer-Verlag; 2005.

[19] Oizumi W, Garcia A, d S Sousa L, Cafeo B, Zhao Y. Code Anomalies Flock Together: Exploring Code Anomaly Agglomerations for Locating Design Problems. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE); 2016. p. 440–451.

[20] Palomba F, Bavota G, Penta MD, Oliveto R, Lucia AD. Do They Really Smell Bad? A Study on Developers' Perception of Bad Code Smells. In: 2014 IEEE International Conference on Software Maintenance and Evolution; 2014. p. 101–110.

[21] d P Sobrinho EV, De Lucia A, d A Maia M. A systematic literature review on bad smells — 5 W's: which, when, what, who, where. IEEE Transactions on Software Engineering. 2018;p. 1–1.

[22] Walter B, Fontana FA, Ferme V. Code smells and their collocations: A large-scale experiment on open-source systems. J Syst Software. 2018;144:1–21.

[23] Pietrzak B, Walter B. Leveraging code smell detection with inter-smell relations. In: International Conference on Ex. P. and Agile Processes in Software Engineering; 2006. p. 75–84.

[24] Yamashita A, Zanoni M, Fontana FA, Walter B. Inter-smell relations in industrial and open source systems: A replication and comparative analysis. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME); 2015. p. 121–130.

[25] Palomba F, Oliveto R, De Lucia A. Investigating code smell co-occurrences using association rule learning: A replicated study. In: 2017 IEEE Work. on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE); 2017. p. 8–13.

[26] Oizumi W, Sousa L, Oliveira A, Garcia A, Agbachi AB, Oliveira R, et al. On the identification of design problems in stinky code: experiences and tool support. Journal of the Brazilian Computer Society. 2018;24(1):13.

[27] Palomba F, Bavota G, Di Penta M, Fasano F, Oliveto R, De Lucia A. A large-scale empirical study on the lifecycle of code smell co-occurrences. Information and Software Technology. 2018;99:1–10.

[28] Lozano A, Mens K, Portugal J. Analyzing Code Evolution to Uncover Relations between Bad Smells; 2015. .

[29] Arcoverde R, Garcia A, Figueiredo E. Understanding the Longevity of Code Smells: Preliminary Results of an Explanatory Survey. In: Proceedings of the 4th

Workshop on Refactoring Tools. WRT '11. Association for Computing Machinery; 2011. p. 33–36.

[30] Yamashita A, Moonen L. Exploring the impact of inter-smell relations on software maintainability: An empirical study. In: 2013 35th International Conference on Software Engineering (ICSE); 2013. p. 682–691.

[31] Oizumi WN, Garcia AF, Colanzi TE, Ferreira M, Staa AV. On the relationship of code-anomaly agglomerations and architectural problems. Journal of Software Engineering Research and Development. 2015;3(1):11.

[32] Martins J, Bezerra C, Uchôa A, Garcia A. Are Code Smell Co-Occurrences Harmful to Internal Quality Attributes? A Mixed-Method Study. In: Proceedings of the 34th Brazilian Symposium on Software Engineering. SBES '20. Association for Computing Machinery; 2020. p. 52–61.

[33] Santana A, Cruz D, Figueiredo E. In: An Exploratory Study on the Identification and Evaluation of Bad Smell Agglomerations. Association for Computing Machinery; 2021. p. 1289–1297.

[34] Tempero E, Anslow C, Dietrich J, Han T, Li J, Lumpe M, et al. Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In: 2010 Asia Pacific Software Engineering Conference (APSEC2010); 2010. p. 336–345.

[35] Sheskin DJ. Handbook of parametric and nonparametric statistical procedures. crc Press; 2020.

[36] Mining HJKMD. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers; 2001.

[37] : Available from: https://github.com/amandads/AgglomerationsExtensionProject.

[38] Brown C, Parnin C. In: Understanding the Impact of GitHub Suggested Changes on Recommendations between Developers. Association for Computing Machinery; 2020. p. 1065–1076.

[39] Terra R, Miranda LF, Valente MT, Bigonha RS. Qualitas.class Corpus: A Compiled Version of the Qualitas Corpus. Software Engineering Notes. 2013;38(5):1–4.

[40] Mantyla MV, Vanhanen J, Lassenius C. Bad smells - humans as code critics. In: 20th IEEE International Conference on Software Maintenance; 2004. p. 399–408.

[41] Fernandes E, Oliveira J, Vale G, Paiva T, Figueiredo E. A Review-Based Comparative Study of Bad Smell Detection Tools. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. EASE '16; 2016. .

[42] Moha N, Guéhéneuc YG. Decor: a tool for the detection of design defects. In: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering; 2007. p. 527–528.

[43] Cruz D, Santana A, Figueiredo E. Detecting Bad Smells with Machine Learning Algorithms: An Empirical Study. In: Proceedings of the 3rd International Conference on Technical Debt. TechDebt '20. Association for Computing Machinery; 2020. p. 31–40.

[44] d F Carneiro G, Silva M, Mara L, Figueiredo E, Sant'Anna C, Garcia A, et al. Identifying Code Smells with Multiple Concern Views. In: 2010 Brazilian Symposium on Software Engineering; 2010. p. 128–137.

[45] Di Nucci D, Palomba F, Tamburri DA, Serebrenik A, De Lucia A. Detecting code smells using machine learning techniques: are we there yet? In: Int'l Conference on Software Analysis, Evolution and Reengineering (SANER); 2018. p. 612–621.

[46] Palomba F, Bavota G, Di Penta M, Oliveto R, De Lucia A, Poshyvanyk D. Detecting bad smells in source code using change history information. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE); 2013. p. 268–278.

[47] Palomba F, Panichella A, De Lucia A, Oliveto R, Zaidman A. A textual-based technique for smell detection. In: 2016 IEEE 24th international conference on program comprehension (ICPC). IEEE; 2016. p. 1–10.

[48] Ichtsis A, Mittas N, Ampatzoglou A, Chatzigeorgiou A. Merging Smell Detectors: Evidence on the Agreement of Multiple Tools. In: Proceedings of the International Conference on Technical Debt. TechDebt '22. Association for Computing Machinery; 2022. p. 61–65.

[49] Alkharabsheh K, Crespo Y, Fernández-Delgado M, Viqueira JR, Taboada JA. Exploratory Study of the Impact of Project Domain and Size Category on the Detection of the God Class Design Smell. Software Quality Journal. 2021 jun;29(2):197–237. https://doi.org/10.1007/s11219-021-09550-5.

[50] Moha N, Gueheneuc YG, Duchien L, Le Meur AF. DECOR: A Method for the Specification and Detection of Code and Design Smells. IEEE Transactions on Software Engineering. 2010;36(1):20–36. https://doi.org/10.1109/TSE.2009.50.

[51] Fokaefs M, Tsantalis N, Stroulia E, Chatzigeorgiou A. JDeodorant: identification and application of extract class refactorings. In: 2011 33rd International Conference on Software Engineering (ICSE); 2011. p. 1037–1039.

[52] Vidal S, Vazquez H, Diaz-Pace JA, Marcos C, Garcia A, Oizumi W. JSpIRIT: a flexible tool for the analysis of code smells. In: 2015 34th International Conference of the Chilean Computer Science Society (SCCC); 2015. p. 1–6.

[53] PMD.: PMD Source Code Analyser. Available from: https://pmd.github.io/.

[54] Tufféry S. Data mining and statistics for decision making. John Wiley & Sons; 2011.

[55] Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of data; 1993. p. 207–216.

[56] Fontana FA, Ferme V, Marino A, Walter B, Martenka P. Investigating the Impact of Code Smells on System's Quality: An Empirical Study on Systems of Different Application Domains. In: 2013 IEEE International Conference on Software Maintenance; 2013. p. 260–269.

[57] Guo Y, Seaman C, Zazworka N, Shull F. Domain-specific tailoring of code smells: an empirical study. In: 2010 ACM/IEEE 32nd International Conference on Software Engineering. vol. 2; 2010. p. 167–170.

[58] Fontana FA, Ferme V, Spinelli S. Investigating the impact of code smells debt on quality code evaluation. In: 2012 Third International Workshop on Managing Technical Debt (MTD); 2012. p. 15–22.

[59] Chidamber SR, Kemerer CF. A metrics suite for object oriented design. IEEE Transactions on Software Engineering. 1994;20(6):476–493.

[60] AlOmar EA, Mkaouer MW, Ouni A, Kessentini M. Do design metrics capture developers perception of quality? an empirical study on self-affirmed refactoring activities. 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). 2019;.

[61] Aniche M.: Java code metrics calculator (CK). Available in https://github.com/mauricioaniche/ck/.

[62] McCabe TJ. A Complexity Measure. IEEE Transactions on Software Engineering. 1976;SE-2(4):308–320. https://doi.org/10.1109/TSE.1976.233837.

[63] Henderson-Sellers B, Constantine LL, Graham IM. Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design). Object Oriented Systems. 1996;3:143–158.

[64] Vargha A, Delaney HD. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics. 2000;25(2):101–132.

[65] Tsay J, Dabbish L, Herbsleb J. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In: Proceedings of the 36th International Conference on Software Engineering. ICSE 2014. Association for Computing

Machinery; 2014. p. 356–366.

[66] Han X, Tahir A, Liang P, Counsell S, Luo Y.: Understanding Code Smell Detection via Code Review: A Study of the OpenStack Community. arXiv.

[67] Digkas G, Ampatzoglou A, Chatzigeorgiou A, Avgeriou P. The Temporality of Technical Debt Introduction on New Code and Confounding Factors. Software Quality Journal. 2022 jun;30(2):283–305. https://doi.org/10.1007/s11219-021-09569-8.

[68] Alizadeh V, Ouali MA, Kessentini M, Chater M. RefBot: Intelligent software refactoring bot. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE; 2019. p. 823–834.

[69] Soares V, Oliveira A, Pereira JA, Bibano AC, Garcia A, Farah PR, et al. On the relation between complexity, explicitness, effectiveness of refactorings and non-functional concerns. In: Proceedings of the 34th Brazilian Symposium on Software Engineering; 2020. p. 788–797.

[70] Silva D, Tsantalis N, Valente MT. Why we refactor? confessions of github contributors. In: Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering; 2016. p. 858–870.

[71] Wohlin C. Experimentation in software engineering. Springer; 2012. Available from: http://gen.lib.rus.ec/book/index.php?md5=5925A42B5016AA2DEC77E9E18CC9648F.

[72] da S Carvalho L, Novais R, Mendonça M. Investigating the Relationship between Code Smell Agglomerations and Architectural Concerns: Similarities and Dissimilarities from Distributed, Service-Oriented, and Mobile Systems. In: Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse. SBCARS '18; 2018. p. 3–12.

[73] Fontana FA, Lenarduzzi V, Roveda R, Taibi D. Are architectural smells independent from code smells? An empirical study. Journal of Systems and Software. 2019;154:139–156.

[74] Hamdi O, Ouni A, AlOmar EA, Mkaouer MW. An Empirical Study on Code Smells Co-occurrences in Android Applications. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW); 2021. p. 26–33.

[75] Vidal S, Oizumi W, Garcia A, Pace AD, Marcos C. Ranking architecturally critical agglomerations of code smells. Science of Computer Programming. 2019;182:64–85.