An Empirical Investigation into the Capabilities of Anomaly Detection Approaches for Test Smell Detection

Valeria Pontillo,[1] Luana Martins,[2] Ivan Machado,[3] Fabio Palomba,[2] Filomena Ferrucci[2]

[1] *Software Languages (Soft) Lab — Vrije Universiteit Brussel, Belgium*
valeria.pontillo@vub.be
[2]*Software Engineering (SeSa) Lab - Department of Computer Science, University of Salerno, Italy*
lalmeidamartins@unisa.it, fpalomba@unisa.it, fferrucci@unisa.it
[3]*Federal University of Bahia, Salvador, Brazil*
ivan.machado@ufba.br

Abstract

Test smells are symptoms of sub-optimal design choices adopted when developing test cases. Previous research has demonstrated their harmfulness for test code maintainability and effectiveness, showing their impact on test code quality. As such, the quality of test cases affected by test smells is likely to deviate significantly from the quality of test cases not affected by any smell and might be classified as *anomalies*. In this paper, we challenge this observation by experimenting with three anomaly detection approaches based on machine learning, cluster analysis, and statistics to understand their effectiveness for the detection of four test smells, i.e., *Eager Test*, *Mystery Guest*, *Resource Optimism*, and *Test Redundancy* on 66 open-source Java projects. In addition, we compare our results with state-of-the-art heuristic-based and machine learning-based baselines. Our ultimate goal is not to prove that anomaly detection methods are better than existing approaches, but to objectively assess their effectiveness in this domain. The key findings of the study show that the *F-Measure* of anomaly detectors never exceeds 47%, obtained in the *Eager Test* detection using the statistical approach, while the *Recall* is generally higher for the statistical and clustering approaches. Nevertheless, the anomaly detection approaches have a higher *Recall* than the heuristic and machine learning-based techniques for all test smells. The low *F-Measure* values we observed for anomaly detectors provide valuable insights into the current limitations of anomaly detection in this context. We conclude our study by elaborating on and discussing the reasons behind these negative results through qualitative investigations. Our analysis shows that the detection of test smells could depend on the approach exploited, suggesting the feasibility of developing a meta-approach.

Keywords: Test Smells, Anomaly Detection, Empirical Software Engineering.

1. Introduction

In the ever-evolving landscape of software development, maintaining high code quality standards is paramount. Despite this, developers introduce defects or flaws during development or evolutionary activities [1]. However, one crucial aspect of software maintenance and evolution is the early detection and resolution of anomalies within the software. Anomalies, in the context of software development, are deviations from expected or normative behavior [2]. These deviations can manifest as code defects [3–5], security vulnerabilities [6, 7], or, as we explore in this study, test smells.

Test smells are symptoms of poor design or implementation choices in test code [8]. Several studies have shown the negative effects of test smells on test code maintainability and effectiveness [9–13], by showing evidence of the risks associated with the presence of test smells for software dependability and test code quality. As a result, researchers have been investigating methods and tools to detect test smells automatically [14–16]. Most of these tools employ heuristic- or rule-based approaches that rely on predefined thresholds to distinguish between non-smelly and smelly test cases [17]. However, these thresholds can be subjective, affecting both the detection capabilities and the trust of developers in using the tools [18]. The limitations of heuristic approaches do not arise from a fundamental inability to detect test smells, but rather from the way these heuristics are defined and implemented. In their current state, the predefined rules and thresholds may not be flexible or precise enough to capture the full spectrum of test smells, leading to challenges that reduce the overall effectiveness of these methods.

To overcome the limitation of having thresholds, recent studies explored machine learning-based methods for test smell detection [19–21]. In addition, prior studies have provided empirical evidence that test smells impact the quality of a test suite [11, 22]. Consequently, we hypothesize that the quality of test cases affected by test smells significantly deviates from the quality of test cases not affected by any smell, allowing us to consider test smells as either collective or point anomalies.

This paper builds on this hypothesis by exploring the use of anomaly detection techniques to identify test smells. Specifically, anomaly detection focuses on identifying deviations from established norms, aiming to uncover patterns that differ significantly from the norm. Given the prior evidence on test smells [11, 22], we argue that test cases impacted by test smells exhibit notable deviations in quality, providing a strong justification for applying anomaly detection in this context. These techniques represent a compelling alternative to traditional supervised machine learning models because they do not require extensive training, thus avoiding the risk of learning from unbalanced datasets [5, 23].

Hence, this paper presents an empirical investigation into the effectiveness of anomaly detection techniques for identifying four test smells across 66 open-source Java projects: *Eager Test*, *Mystery Guest*, *Resource Optimism*, and *Test Redundancy*. Our goal is to investigate how well anomaly detection methods perform when applied to the specific task of test smell detection, as opposed to demonstrating their superiority with respect to existing approaches. In other terms, our intention is not to prove that these methods are better than others but to objectively assess their effectiveness in this domain. To this aim, we first evaluate the performance of three different anomaly detection techniques, i.e., machine learning-based, cluster algorithm-based, and statistical-based approaches. Then, we compare these approaches against state-of-the-art test smell detectors that use heuristics and machine learning models.

The results of our study reveal several key findings. First, we observe that, in a cross-project scenario, the three approaches for anomaly detection do not perform well in detecting any of the four test smells, as no technique achieved an *F-Measure* higher than 47%. Moreover, the anomaly detection approaches did perform better than the state-of-the-art baselines, especially regarding *Recall*. An additional overlap analysis indicates that test smells detection may depend on the specific approach used, suggesting the potential for developing a meta-approach. Lastly, our qualitative investigation identifies areas for refinement, which could lead to the development of more effective heuristics or meta-approaches in the future.

All in all, our study is the first to investigate test smells from the perspective of code anomalies. This novel approach introduces a new viewpoint to the field and lays the groundwork for future research, regardless of whether the results are positive or negative. Even though the performance metrics were lower than expected, we are confident that our findings provide valuable insights into the challenges of using anomaly detection for test smell detection. These insights can guide future studies and contribute to the refinement and improvement of both anomaly detection techniques and test smell detection methodologies, further advancing research in this area.

To sum up, this paper provides the following contributions:

1. An empirical investigation into the capabilities of anomaly detection techniques for test smell detection;

2. A set of insights to better contextualize our results and delineate future research direction on the matter;

3. An online appendix [24] providing all data and scripts used to conduct our study that the research community can use to replicate and build upon our empirical study.

Structure of the paper. Section 2 covers background and related work. Section 3 presents the research questions, followed by data collection and analysis methods in Section 4. Findings are discussed in Section 5, with implications in Section 6. Section 7 addresses threats to validity, and Section 8 concludes the paper and outlines future research directions.

2. Background and Related Work

This section provides background information and references to related work on anomaly detection and test smells, finally discussing how test smells can be classified as anomalies.

2.1. Anomaly Detection

Anomaly detection is the process of identifying uncommon data instances that raise suspicions due to their substantial deviation from the majority of the data [2]. Anomaly detection finds extensive use in a wide variety of applications such as fraud detection for credit cards [25], disease diagnosis [26, 27], intrusion detection for cyber-security [6, 7], fault detection in software systems [3–5].

A straightforward anomaly detection approach is to define a region that encapsulates normal behavior and classify any data point outside of this region as an anomaly [2]. However, this approach becomes challenging given various factors, such as the nature of the input data, the notion of normal behavior for different application domains, and the availability of labeled data that might contain noise similar to the actual anomalies [2, 7].

According to Chandola et al. [2], the starting point for the formulation of the detection anomaly is the understanding of the nature of the data input. The data input is a collection of data instances, and each data instance might consist of only one attribute (univariate) or multiple attributes (multivariate) of different types (binary, categorical, or continuous). Another key point is the nature of the anomaly. A point anomaly is a particular data instance that deviates from the normal pattern of the dataset. A contextual anomaly has been most commonly explored in time-series data, being classified as a data instance that behaves anomalously in a particular context. A collective anomaly is a collection of similar data instances that behave anomalously with respect to the entire dataset, i.e., the co-occurrence of data instances is an anomaly.

Another key point is the association of labels to the data input to denote whether that instance is normal or anomalous. Based on the extent to which the labels are available, three anomaly detection techniques can be applied. Supervised anomaly detection requires training the dataset that has labeled instances for normal and anomaly classes. Semi-supervised anomaly detection assumes that the training data has labeled instances only for the normal class. Unsupervised Anomaly Detection creates groups of data based on density or distance thresholds instead of labeled data. Due to limited access to anomalous data, constructing an anomaly detector is often conducted under semi- or unsupervised techniques [2, 28, 29].

Lastly, the anomaly detection techniques produce either binary or score-based outputs. The former classifies the data instances in a binary fashion, i.e., either anomalous or normal.

The latter assigns a score to each data instance, which is used as a threshold to select which of them are anomalies [2, 28, 29].

2.2. Test Smells

Test smells are symptoms of poor design or implementation choices in test code [8]. Several studies have investigated the negative effects associated with the existence of test smells in the test suites. Bavota et al. [9, 10] performed two key empirical studies to investigate the diffusion and effects of test smells on software maintenance. Palomba et al. [30] analyzed the diffusion of test smells on automatically generated test suites. Spadini et al. [11] focused on how bad practices affect test code quality, investigating ten open-source projects to find a relation between six test smells and the change and defect-proneness of both test and production code. Peruma et al. [31] performed an empirical study on the distribution and survivability of test smells. Martins et al. [32] conducted an empirical study to investigate the diffusion of test smells and their relationship with structural metrics. Those consistently support each other's findings, demonstrating that test smells are highly diffused in test classes, and their presence can reduce test code comprehension compared to the absence of test smells.

On that account, researchers have dedicated efforts to propose methods and automated tools to detect test smells [14–16]. Aljedaani et al. [17] listed 22 peer-reviewed tools for detecting 66 test smells. Most tools rely on rules and metrics-based approaches to detect test smells in test codes written with the JUnit framework. Some tools focus on a specific category of test smells, such as redundant tests (TeReDetect [14] and TeCReVis [33]), dependent tests (DTDetector [34], ElectricTest [35], TEDD [36], PraDeT [37], and PolDet [38]), and test fixtures (TestHound [39] and TestEvoHound [40]). Other tools focus on different categories of test smells that affect different parts of the test code (OraclePolish [41], Taste [42], tsDetect [15], JNose Test [16]). In addition, there are tools for the detection of test smells and calculation of test code metrics (VITRuM [43] and JTDog [44]). Yang et al. [45] investigated new test smell types from software practitioners' discussions and developed a new tool to detect them.

Complementary, there are some approaches based on machine learning to detect test smells. Pontillo et al. [21] proposed a test smell detection approach based on machine learning to detect four test smells using textual and structural metrics. Martins et al. [19] used structural metrics of test code to train machine learning algorithms for classifying four test smells. Hadj-Kacem and Bouassida [20] have analyzed the agreement level among the detection tools and suggested a multi-label classification approach to detect test smells.

3. Research Question and Variables

The *goal* of the empirical study was to evaluate the performance of three different anomaly detection approaches, i.e., machine learning-based, cluster algorithm-based, and statistical-based approaches, when employed for test smell detection, with the *purpose* of understanding how these approaches work and how they compare to baseline techniques based on heuristics and machine learning. The *perspective* is from both researchers and practitioners: the former are interested in assessing the feasibility of using anomaly detection for test smell prediction; the latter are interested in evaluating the feasibility of using those methods within their contexts. Specifically, as first step, we applied various anomaly detection approaches for identifying four specific test smells within software projects and evaluated the resulting performance:

Table 1: Definition of the test smells included in our study.

Test Smell	Definition
Eager Test	A test method involving many methods of the object being tested.
Mystery Guest	A test that uses external resources (e.g., databases or files).
Resource Optimism	A test that uses external resources without checking the state of these.
Test Redundancy	A test that could be removed without impacting the test suite.

> **RQ$_1$**
>
> *To what extent can anomaly detection detect test smells?*

Since we analyzed three different anomaly detection approaches that can impact on test smell detection differently, we considered three sub-research questions:

RQ$_{1.1}$. *To what extent can machine learning-based anomaly detection detect test smells?*

RQ$_{1.2}$. *To what extent can cluster algorithm-based anomaly detection detect test smells?*

RQ$_{1.3}$. *To what extent can statistical-based anomaly detection detect test smells?*

Upon assessing the performances of anomaly detection methods, we compared them against state-of-the-art baseline approaches based on heuristics and machine learning. Hence, we formulated our second research question:

> **RQ$_2$**
>
> *How do anomaly detection perform compared to heuristic- and machine learning-based approaches?*

Figure 1 overviews the empirical study setting, which consisted of three main steps: (1) Context Selection, (2) Detection of test smells using anomaly detection approaches - **RQ**$_1$, (3) Detection of test smells detection using the baseline techniques - **RQ**$_2$, and (4) Data analysis - **RQ**$_1$ and **RQ**$_2$. To design and report our empirical study, we followed the empirical software engineering guidelines by Wohlin et al. [46] and the *ACM/SIGSOFT Empirical Standards*.[1]

[1] Available at `https://github.com/acmsigsoft/EmpiricalStandards`.

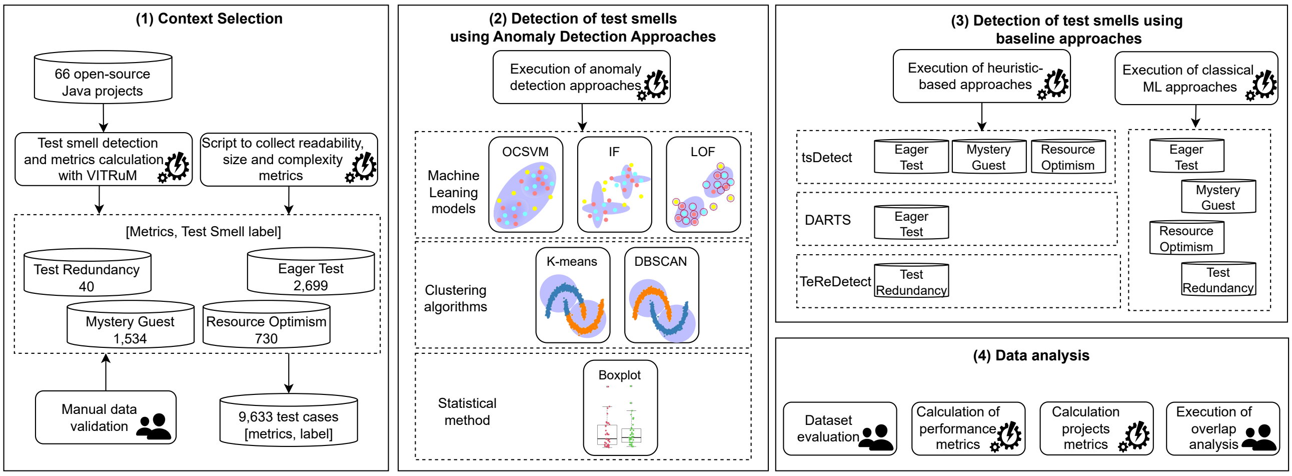
(1) Context Selection
66 open-source Java projects
Test smell detection and metrics calculation with VITRuM
Script to collect readability, size and complexity metrics
[Metrics, Test Smell label]
Test Redundancy 40
Eager Test 2,699
Mystery Guest 1,534
Resource Optimism 730
Manual data validation
9,633 test cases [metrics, label]
(2) Detection of test smells using Anomaly Detection Approaches
Execution of anomaly detection approaches
Machine Leaning models
OCSVM
IF
LOF
Clustering algorithms
K-means
DBSCAN
Statistical method
Boxplot
(3) Detection of test smells using baseline approaches
Execution of heuristic-based approaches
Execution of classical ML approaches
tsDetect
Eager Test
Mystery Guest
Resource Optimism
DARTS
Eager Test
TeReDetect
Test Redundancy
Eager Test
Mystery Guest
Resource Optimism
Test Redundancy
(4) Data analysis
Dataset evaluation
Calculation of performance metrics
Calculation projects metrics
Execution of overlap analysis

Figure 1: Overview of the research method employed.

3.1. Context Selection

The *context* of our study consisted of a dataset previously collected [21] by analyzing and manually validating test cases from 66 open-source Java projects. This dataset contains information on test smells whose detection is based on at least two metrics to enable the experimentation with machine learning-based anomaly detection approaches (to answer to $\mathbf{RQ}_{1.1}$), and their detection is supported by at least one tool to address $\mathbf{RQ}_2$. The availability of a manually validated dataset enabled us to perform a more precise evaluation of the approaches without the potential biases introduced by automatically generated data. More particularly, the dataset collects 9,633 test cases. For each test case, the dataset provides manually-validated information regarding the presence of four different test smells: *Eager Test*, *Mystery Guest*, *Resource Optimism*, and *Test Redundancy*. Table 1 reports the definition of these test smells. The total number of smelly test cases is 3,652, which represents 37.9% of the total test cases. The number of smelly test cases does not coincide with the sum of instances across all four test smell types: indeed, a single test case can be affected by multiple test smell instances. This is the case with the dataset exploited. Looking at the distribution of the individual test smell types, we have 2,699 *Eager Test* instances (28%), 1,534 *Mystery Guest* instances (15%), 730 *Resource Optimism* instances (7%), and 40 *Test Redundancy* instances (0.4%). As a consequence, each smelly test case presents an average of 1.37 test smell instances - considering the total number of individual instances (2699+1534+730+40=5,003) over the number of smelly test cases (3,652).

It is also important to remark that our study examines each test smell individually: we indeed addressed anomaly detection methods and baselines specifically for detecting instances of *Eager Test*, *Mystery Guest*, *Resource Optimism*, and *Test Redundancy*. As a consequence, we split the problem of test smell detection into four sub-problems for which the amount of smelly instances is much lower than the global number of test smell instances (*Eager Test*=28%; *Mystery Guest*=15%; *Resource Optimism*=7%; *Test Redundancy*=0.4%).

When considering this perspective, the adoption of anomaly detection methods starts being justified. In addition, it is also worth pointing out that the scope of anomaly detection is centered on identifying deviations from normality, which are not necessarily related to the frequency of occurrences. Indeed, anomaly detection aims to uncover patterns that differ significantly from the norm, regardless of how often they appear. Our work stems from this foundational principle. Based on previous studies on test smells [11, 22], we hypothesize that the quality of test cases affected by test smells significantly deviates from the quality of test cases not affected by any smell, thereby justifying the use of anomaly detection methods.

This dataset represents the largest, most recent, and manually validated set of test smell instances available in the literature. The dataset includes projects with different characteristics, hence allowing us to feature a study of a pretty diverse set of software systems. To provide more demographic information on the dataset, we further analyzed the scope of the projects, mining the descriptions of the projects from the README files of the corresponding repositories. In addition, we computed metrics that may provide insights into their diversity: in particular, we computed the number of stars, forks, open issues, and the number of lines of code. In terms of scope, we observed that the considered projects vary very much, e.g., some projects relate to HTTP requests and responses, others to container orchestration or web frameworks. The number of stars varies between 61 and 62,255, with a median value of 404. In terms of size, the smallest project has 105 lines of code, while the biggest project has 575,872. The median value is 57,719. Finally, the median values for the number of forks and open issues are 1,875 and 103, respectively —a full report of the domains of the considered projects is available in our online appendix [24].

3.2. Dependent Variable

The dependent variable of our study is the presence or absence of each test smell type. Thus, we experimented with the anomaly detection approaches for *Eager Test*, *Mystery Guest*, *Resource Optimism*, and *Test Redundancy* independently.

3.3. Independent Variable

To collect a set of reliable independent variables for each test smell under consideration, we first used the structural and textual metrics employed by state-of-the-art heuristic-based and machine learning-based approaches. These metrics were already available in the literature and used in several detection approaches [15, 47–49], hence representing a valid starting point for characterizing the smelliness of test code. In particular, for *Eager Test*, we analyzed the number of multiple calls made by a test method to multiple production methods and the test method cohesion of each test case. For *Mystery Guest*, we analyzed whether the test case contains instances of external resources, i.e., a file. The third, i.e., *Resource Optimism*, was identified by analyzing whether a test method used an external resource without checking its state. Finally, we used as independent variables for detecting *Test Redundancy* the code coverage to analyze whether two tests cover similar paths.

In an effort to characterize test code quality from alternative perspectives, we also complemented the state-of-the-art metrics with two further categories of test metrics proposed in previous research on test code quality [50–52]. Specifically:

Test code size and complexity. We computed the number of lines of test code. This is a well-established proxy metric for complexity [53] and may estimate the quality of test cases from the perspective of maintainability [54]. Secondly, we computed the McCabe cyclomatic complexity of test code [55], which estimates the logical complexity of tests that, in turn, can affect the likelihood of the emergence of test smells.

Test readability. According to previous studies, readability represents a key aspect for assessing test code quality [56], and it is associated with an increasing amount of test smells [57]. We exploited these findings and enlarged the set of metrics considered with those coming from the paper by Scalabrino et al. [50]. In particular, it proposed a comprehensive set of 67 readability metrics that encompass multiple aspects, including comment readability, expression complexity, method chains, semantic coherence, etc. This metric suite (1) integrates and enlarges the metrics originally defined within the seminal studies on code readability by Buse and Weimer [58], and Dorn [59] and (2) has been assessed in the context of studies on test code quality assessment [13, 57].

On the one hand, the addition of these two categories of metrics may have potentially enhanced the overall detection capabilities of anomaly detection methods. On the other hand, assessing the role of these metrics may possibly offer new, valuable insights into the improvement of existing test smell detectors. Metrics related to test code size, complexity, and readability were computed for all the test smells considered, thus complementing the specific set of metrics tailored to the individual test smells. Overall, our study featured a set of 71 metrics. Our online appendix [24] reports information on all the definitions of the metrics computed in the study.

4. Research Method

This section discusses the research methods employed to address the research questions targeted by our work.

4.1. Data Preparation

Before building the experimented anomaly detectors, we performed data exploration and quality checks, ensuring the suitability of the data for our analysis. In particular, we examined the skewness of the data [60], verified the presence of missing values, and normalized the data using a *standard* scaling. In such a context, the data scaling is useful to (i) ensure that each feature contributes equally to the analysis, (ii) potentially improve the ability to detect anomalies, and (iii) facilitate the interpretation of the results.

As an exemplary case, Table 2 reports the values obtained when computing the skewness of the test smell-specific metrics, i.e., those employed by state-of-the-art heuristic-based approaches. For the sake of readability, we report the skewness of all the remaining metrics as part of the online appendix [24]. We can observe that all metrics except one, i.e., Test Method Cohesion, present high positive skewness with values significantly different from zero. This result suggests that data do not follow a symmetric distribution by indicating the presence of outliers and further reinforces our assumption about considering test smells as *anomalies*. Similar results were observed when considering the test code, complexity, and readability metrics, hence confirming that the dataset actually presents anomalies that may be captured by anomaly detection methods.

Table 2: The skewness for each independent variable of our empirical study.

Test Smell	Metric	Value
Eager Test	NMC	4.06
	TMC	-0.13
Mystery Guest	NRDB	11.03
	NRF	7.69
Resource Optimism	ERNC	10.92
	FRNC	9.03
Test Redundancy	PR	7.23
	SR	8.22

*4.2. **RQ**$_{1.1}$: Research Method*

The first step of the analysis was the definition of the machine learning-based anomaly detection methods. We focused on three models, i.e., *OneClassSVM*, *IsolationForest*, and *LocalOutlierFactor*. We selected these algorithms for two reasons: first, each model is based on a different class of algorithms, allowing us to provide a wider overview of how anomaly detection can be applied to test smell detection. Then, according to Chandola et al. [2], these methods are the most stable and reliable, other than being commonly used in multiple environments [4, 5, 61, 62]. Particularly, the three methods are defined as follows:

OneClassSVM (OCSVM) [63]. This algorithm is based on the Support Vector Machine (SVM). Similarly, the algorithm defines a frontier that delimits the initial observations. The future observations will either fall in the frontier, therefore belonging to the same class as the original data (normal), or they will fall outside the frontier, therefore classifying as anomalous data.

IsolationForest (IF) [64]. This is an *ensemble* technique based on the *Extremely Randomized Tree* model [65], which randomly selects a feature and a split value between the maximum and minimum of the selected feature. The number of splits required to isolate a sample equals the path length from the root to the final node of a tree. Since random partitioning produces noticeably shorter paths for anomalies, when a forest of random trees produces shorter paths for particular samples, these are highly likely to be anomalies.

LocalOutlierFactor (LOF) [66]. The algorithm computes the local density of a sample compared to its neighbors. The density of an observation is given by the ratio of the average local density of its k-nearest neighbors and its own local density. If the density is different from that of its neighbors, the analyzed sample is an outlier; otherwise, it is considered normal.

It is important to emphasize that these algorithms are specifically designed to handle imbalanced datasets because they operate under the assumption that anomalous data points are much fewer compared to normal data points. Therefore, traditional data resampling techniques like SMOTE, which are commonly used in supervised learning scenarios to address class imbalance, are not applicable in the context of anomaly detection-based machine learning methods. In fact, applying such resampling techniques could introduce bias by artificially altering the distribution of the dataset, potentially compromising the accuracy and reliability of the anomaly detection performance.

We experimented with the hyper-parameters of the classifiers using the *Random Search* strategy [67] that randomly samples the hyper-parameters space to find the best combination of hyper-parameters maximizing a scoring metric (i.e., the F-Measure). The models were implemented using the SCIKIT-LEARN library [68]—more details on the implementation and the hyperparameters are reported in our online appendix [24]. To evaluate the performance of the machine learning-based anomaly detection approaches, we computed several state-of-the-art metrics such as *Precision*, *Recall*, *F-Measure* [69], and *Matthews Correlation Coefficient* (*MCC*) [70].

4.3. *RQ*$_{1.2}$: *Research Method*

Chandola et al. [2] defined two categories of clustering-based anomaly detection approaches. The first starts from the assumption that normal data instances belong to a cluster in the data, while anomalies do not belong to any cluster. The second assumes that normal data instances lie close to their closest cluster centroid, while anomalies are far away from their closest cluster centroid. We analyzed one algorithm for each category:

DBSCAN. The Density-Based Spatial Clustering of Applications with Noise (DBSCAN), proposed by Ester et al. [71], works by grouping data that are close to each other. The algorithm requires two parameters, i.e., the *radius* (ϵ) and the *minimum number of points* to define a dense region (min-sample). The algorithm works as follows: first, it chooses a random data point that has not been analyzed yet. Then, the algorithm returns all data within a distance ϵ from the chosen point. If at least min-sample points are within the ϵ distance, the algorithm creates a new cluster and adds all the points. Alternatively, the algorithm marks the chosen point as noise if there are less than min-sample points within the ϵ distance.

K-means. This is a centroid-based clustering algorithm [72] that computes the distance between each data point and a centroid to assign it to a cluster. The K-means algorithm aims to identify the K number of groups in the dataset. K-means strives to minimize the *within-cluster sum of squares* (WCSS), which represents the sum of squared distances between data points and their respective cluster centroids. K-means aims to find K clusters, each with its centroid and assigns each data point to the cluster with the closest centroid.

To determine the optimal value of K, we employed the elbow methods, i.e., a technique that involves the algorithm with different values of K and plotting the corresponding WCSS for each K. The plot exhibits a decreasing trend in WCSS as K increases, with the rate of decrease in WCCS slowing as K grows. The ”elbow point” on the plot, where the WCSS reduction starts to slow down significantly, is often considered the optimal K value. In addition, we applied a threshold based on the standard deviation of the WCSS values. By calculating the standard deviation for the WCSS values across different K values, we identified a threshold beyond which K does not significantly impact the clustering quality. This threshold helps in avoiding overfitting the number of clusters.

Finally, to evaluate the performance of the clustering-based anomaly detection approaches, we computed several state-of-the-art metrics such as *precision*, *recall*, *F-Measure* [69], and *Matthews Correlation Coefficient* (*MCC*) [70].

4.4. *RQ*$_{1.3}$: *Research Method*

Concerning the research method applied to answer to **RQ**$_{1.3}$, we decided to apply the *boxplot rule* as statistical technique to detect the anomalies in our data. A boxplot provides a visual representation of data by highlighting key summary statistics, including the smallest non-anomalous observation (*min*), the lower quartile (*Q1*), the median, the upper quartile (*Q3*), and the largest non-anomalous observation, (*max*). The difference between *Q3* and *Q1* is referred to as the *Interquartile Range (IQR)*. The boxplot also specifies the thresholds beyond which any data point is considered an anomaly. An instance that lies more than 1.5**IQR* lower than *Q1* or 1.5**IQR* higher than *Q3* is declared as an anomaly.

Finally, after applying the statistical-based anomaly detection approach in a cross-project scenario, we computed *precision*, *recall*, *F-Measure* [69], and *MCC* [70].

4.5. *RQ*$_2$: *Research Method*

After addressing the performance of multiple anomaly detection methods in the task of test smell detection, we compared them against state-of-the-art baseline approaches in an effort to understand the strengths and weaknesses of anomaly detection in comparison to alternative detectors. More particularly, we identified two categories of baselines. In the first place, we experimented with heuristic-based detectors: these approaches make use of code metrics and predefined thresholds to discriminate the smelliness of test code. In the second place, we exploited canonical machine learning models: these approaches leverage labeled datasets to train algorithms that can learn to identify test smells by recognizing complex patterns and relationships within the training data.

As for the heuristic-based detectors, we selected three approaches proposed in previous work. Specifically:

tsDetect [15]. We selected this tool as it represents the current state of the art in test smell detection [17]. This tool is able to detect three of the test smells included in our study, i.e., *Eager Test*, *Mystery Guest*, and *Resource Optimism*. In particular, the first is detected by computing the number of the multiple calls made by a test method to multiple production methods. The second is identified by analyzing whether a test method contains instances of files and database classes. Finally, the third is identified by looking at whether a test method utilizes a `File` instance without calling the method `exists()`, `isFile()`, or `notExist()`. The detection rules previously described are inspired by the original definition of the smells provided by Van Deursen et al. [8]. Other available tools, such as JNose Test [16] and TestQ [73], employ exactly the same detection mechanism, hence selecting TsDetect ensured that we covered the essential detection rules shared by multiple widely recognized tools.

TeReDetect [14]. We selected this tool as it is the only one, along with TeCReVis [33], to detect instances of the *Test Redundancy* smell. The tool detects the smell by computing code coverage and analyzing whether two tests cover similar paths. While both tools implement the same detection rules, TeCReVis presents the results through a user interface [17].

Darts [48]. The tool relies on the detection rule proposed in TASTE by Palomba et al. [42]. These tools differ from the previous ones as they do not rely on traditional code metrics but instead implement heuristics based on textual similarity metrics. Specifically, they detect *Eager Test* instances through a two-step process: first, the test method calls are replaced with the actual production code methods called by the test method; then, the conceptual cohesion metric is computed, taking into account the constituent methods and, whether this metric exceeds 0.5 the smell is detected. Among these, we selected DARTS, as it is the actual implementation of the TASTE approach and thus provided a concrete tool.

These baselines were selected based on (1) their capabilities to detect the test smells considered in our work; and (2) their demonstrated detection performance, which was shown to be sufficiently high in the original research papers that introduced them. By selecting these well-established baselines, we were able to comprehensively evaluate the effectiveness of anomaly detection methods in comparison to the current state-of-the-art heuristic-based approaches for test smell detection.

As for the machine learning-based detectors, the selection process exploited recent findings in the field. Drawing from previous work in the area [21], which comprehensively evaluated the performance of supervised machine learning models for test smell detection against the same dataset as our study, we selected the best-performing machine learning model for each test smell considered. In particular, the previous study [21] trained and evaluated six supervised machine learning algorithms in both within- and cross-project scenarios for the four test smells under analysis. The results indicated that, for the cross-project scenario, the best machine learning model for *Eager Test* detection is AdaBoost with NearMiss2 as the balancing technique; for *Mystery Guest* detection is Random Forest with NearMiss1; for *Resource Optimism* detection is Support Vector Machines with random undersampling; and for *Test Redundancy* detection is Naive Bayes with no balancing. These four models served as additional baselines, allowing us to compare their performance against both anomaly detection and heuristic-based approaches.

It is important to emphasize that none of the heuristic-based detectors required additional configuration. They were executed directly on the source code without the need for parameter adjustments, ensuring the use of their original implementations and avoiding any potential bias from incorrect configurations. Similarly, for the machine learning models, we followed the same configuration pipeline as in the prior study [21].

To enable the comparison, we employed the same evaluation metrics used to assess the anomaly detection approaches, i.e., *precision*, *recall*, *F-Measure*, and *MCC*.

4.6. *RQ*$_1$-*RQ*$_2$: *Statistical Analysis*

In addressing both research questions of the study, we conducted a statistical analysis of the results to determine whether the observed performance differences among the experimented approaches were statistically significant. This analysis aimed to validate the reliability and robustness of the findings.

More specifically, the statistical analysis was implemented by combining the application of the Friedman [74] and Nemenyi [75] tests. The Friedman test is a non-parametric test used to detect differences in treatments across multiple test attempts, which is particularly useful when the assumptions of normality are not met. The Nemenyi test is a post-hoc analysis performed after the Friedman test to identify which groups differ from each other. The statistical tests were selected based on the analysis of our data assumptions. Specifically, we verified normality with the Shapiro-Wilk test [76] and found that the data was not normally distributed (p-value<0.05). As such, we opted for the Friedman test [74], ensuring a random sample from each group, with at least three different observations per group, and a continuous dependent variable. For multiple testing corrections, we applied the Nemenyi post-hoc test [75], which adjusts for multiple comparisons and controls the family-wise error rate. In the context of **RQ**$_1$, the tests were applied to determine if there were any statistically significant differences in the *F-Measure* scores of the six anomaly detection methods experimented with. For **RQ**$_2$, the tests were applied to verify the differences between the *F-Measure* scores achieved by the anomaly detection methods and baseline approaches.

5. Analysis of the Results

This section provides an overview of the results for each research question and sub-research question.

5.1. Machine Learning-based Anomaly Detection

In the context of **RQ**$_{1.1}$, we aimed to assess the performance of machine learning-based anomaly detection methods for test smell detection. Figure 2 depicts boxplots reporting the distribution of the *F-Measure* obtained by the experimented techniques over the considered test smells. We can observe that for all test smells, the performance is low, i.e., the median is zero or close to zero for all models. For this reason, the machine learning-based anomaly detection method does not seem useful for test smell detection.

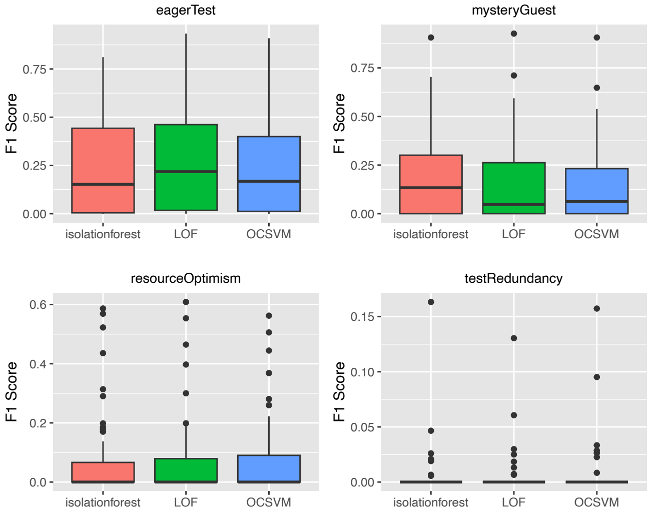

Figure 2: Boxplot representing the F-Measure values obtained by classifiers for all considered test smells.

Concerning the other performance metrics, Tables 3, 4, and 5 present the performance results respectively for *Isolation Forest*, *Local Outlier Factor*, and *One-Class SVM*. We can observe that the performances are generally low when the algorithms are run without hyper-parameter optimization, with the *MCC* ranging from -30% to 1%. Furthermore, regarding the *F-Measure*, the values span from 0% to 34% with the lowest value observed for *Test Redundancy* and the highest for *Eager Test*, thus confirming the previous analysis based on boxplots. As for *Precision*, the performance varies from 0% to 18%, while the highest *Recall* is 75% for *Test Redundancy*.

Overall, we found that the hyper-parameter optimization did not improve the performance as much as to justify the high computational cost required.

5.2. Cluster algorithm-based Anomaly Detection

In the context of **RQ**$_{1.2}$, we aimed to assess the performance of cluster algorithm-based anomaly detection methods for test smell detection. Figure 3 shows the distribution of metrics values in clusters for all test smells using DBSCAN. *Cluster -1* (royal blue) indicates data points that do not belong to any of the identified clusters and, therefore, are considered anomalies.

The DBSCAN algorithm generated 12 clusters for *Eager Test*, 19 for *Mystery Guest*, 16 for *Resource Optimism*, and seven for *Test Redundancy*. With respect to *Eager Test*, as the NMC (Number of Methods Call) values increase, the dispersion of TMC (Test Method Cohesion) values tends to decrease. On the one hand, *Cluster 0* and Cluster 13 have test methods with one call for production methods ($NMC = 1$) and high cohesion values (TMC has $\bar{x} = 0.82$ for Cluster 0 and $\bar{x} = 0.98$ for Cluster 13). The difference between the clusters lies in the Dorn metrics; most are higher in Cluster 13. On the other hand, the test methods in other clusters have 0 calls for production methods ($NMC = 0$) and groups them by cohesion range; Cluster 5 has the lowest cohesion ($TMC = 0$), and Cluster 8 has the highest cohesion values ($\bar{x} = 0.08$, $\sigma = 0.01$). With respect to *Mystery Guest*, test methods with no references to external files nor references to databases ($NRF = 0$ and $NRDB = 0$) are grouped according to the number of lines and complexity into Clusters 0 to 16. For example, Cluster 7 has more lines ($tloc = 7$) and less complexity ($tmcCabe = 1$) than the others. Similar reasoning are applied to the *Resource Optimism*. Finally, for *Test Redundancy*, most clusters have no pair redundancy ($PR = 0.0$) nor suite redundancy ($PR = 0.0$). Differently, Cluster 2 has pair redundancy ($\bar{x} = 0.02$ and $\sigma = 0.0$), Cluster 4 has suite redundancy ($\bar{x} = 0.03$, and $\sigma = 0.02$), and Cluster 12 has both pair redundancy ($\bar{x} = 0.81$, and $\sigma = 0.0$) and suite redundancy ($\bar{x} = 0.0$, and $\sigma = 0.16$).

In terms of performance, Table 6 reports the confusion matrix other than the performance metrics. We can observe that the *F-Measure* ranges between 0% to 43% (the highest value is for *Eager Test*). The highest *MCC* reached is 2% for the *Resource Optimism* detection. Finally, the *Precision* is low (the highest value is 27% for *Eager Test*), while the *Recall* is high for all test smells, ranging between 95% and 100%.

Turning our attention to the K-means algorithm, Figure 4 shows the distribution of metrics values in two clusters for all test smells, i.e., the optimal value of K according to the elbow method. *Cluster 1* (pastel lavender) indicates data points considered as anomalies.

With respect to *Eager Test*, Cluster 0 has lower median values for NMC than Cluster 1 ($\bar{x} = 1.18$, and $\bar{x} = 1.99$) and higher median values for TMC than Cluster 1 ($\bar{x} = 0.21$, and $\bar{x} = 0.17$). For *Mystery Guest*, Cluster 0 has lower median values for NRDB ($\bar{x} = 0.18$, and $\bar{x} = 0.75$) and NRF ($\bar{x} = 0.54$, and $\bar{x} = 3.34$) than Cluster 1. Differently, for *Resource Optimism*, Cluster 0 has higher median values for ERNC ($\bar{x} = 0.18$, and $\bar{x} = 0.75$) and FRNC ($\bar{x} = 0.74$, and $\bar{x} = 2.49$) than Cluster 1.

Table 3: Aggregate results for *Isolation Forest* in terms of *Precision*, *Recall*, *Accuracy*, *F-Measure*, and *MCC* without (i.e., "w/o HT") and with (i.e., "w/ HT") the hyper-parameter optimization by Random Search.

	Precision		**Recall**		**Accuracy**		**F-Measure**		***MCC***	
Test Smell	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**
Eager Test	0.18	0.25	0.43	0.83	0.30	0.26	0.25	0.38	-0.30	-0.19
Mystery Guest	0.14	0.15	0.60	0.87	0.36	0.19	0.23	0.25	-0.06	-0.08
Resource Optimism	0.07	0.07	0.61	0.88	0.35	0.14	0.12	0.13	-0.03	-0.03
Test Redundancy	0.004	0.004	0.75	0.92	0.33	0.10	0.01	0.01	0.01	0.005

Table 4: Aggregate results for *Local Outlier Factor* in terms of *Precision*, *Recall*, *Accuracy*, *F-Measure*, and *MCC* without (i.e., "w/o HT") and with (i.e., "w/ HT") the hyper-parameter optimization by Random Search.

	Precision		**Recall**		**Accuracy**		**F-Measure**		***MCC***	
Test Smell	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**
Eager Test	0.25	0.25	0.54	0.54	0.43	0.43	0.34	0.34	-0.07	-0.07
Mystery Guest	0.15	0.15	0.56	0.56	0.42	0.42	0.23	0.23	-0.03	-0.03
Resource Optimism	0.07	0.07	0.56	0.56	0.41	0.40	0.12	0.12	-0.02	-0.02
Test Redundancy	0.004	0.005	0.70	0.70	0.40	0.40	0.01	0.01	0.01	0.01

Table 5: Aggregate results for *OCSVM* in terms of *Precision*, *Recall*, *Accuracy*, *F-Measure*, and *MCC* without (i.e., "w/o HT") and with (i.e., "w/ HT") the hyper-parameter optimization by Random Search.

	Precision		**Recall**		**Accuracy**		**F-Measure**		***MCC***	
Test Smell	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**	**w/o HT**	**w/ HT**
Eager Test	0.17	0.34	0.28	0.56	0.41	0.58	0.21	0.42	-0.23	0.13
Mystery Guest	0.13	0.18	0.38	0.50	0.50	0.55	0.19	0.26	-0.07	0.04
Resource Optimism	0.06	0.07	0.40	0.49	0.52	0.50	0.11	0.13	-0.04	-0.00
Test Redundancy	0.005	0.005	0.65	0.62	0.52	0.51	0.01	0.01	0.02	0.02

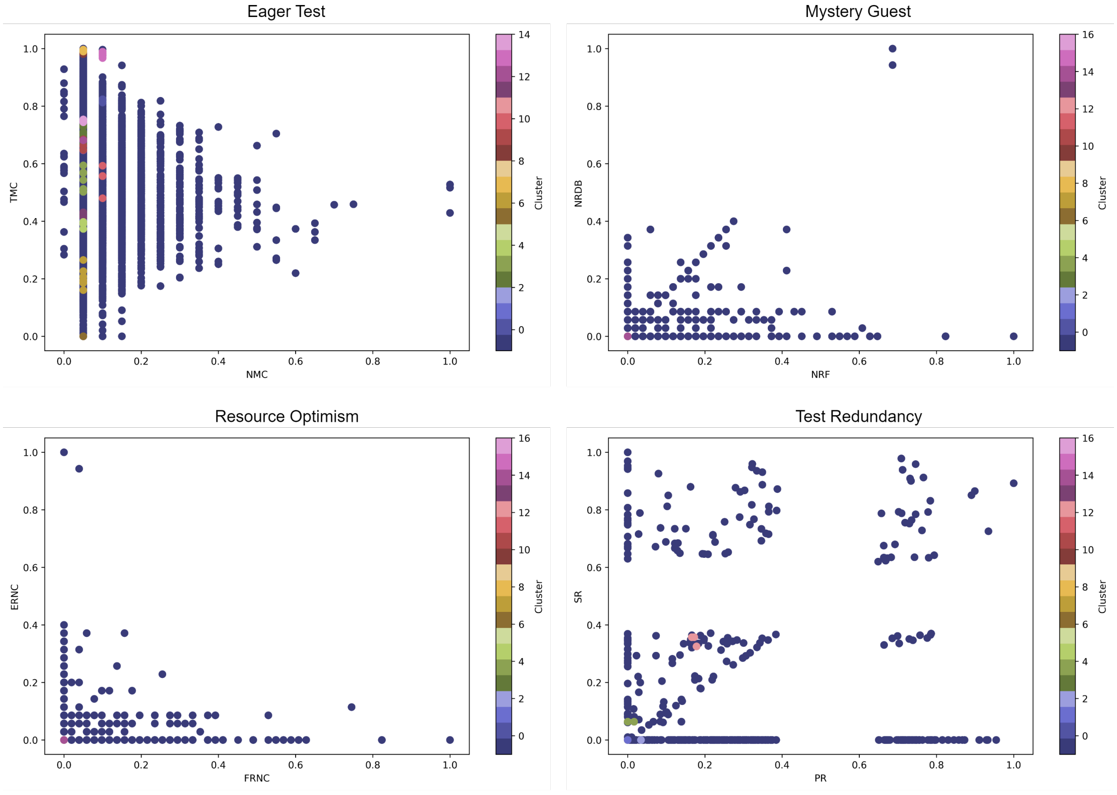

Figure 3: Scatter plot to show the spatial distribution of metrics values in clusters for all test smells using DBSCAN.

Table 6: Results for the cluster-based approach in terms of *Precision*, *Recall*, *Accuracy*, *F-Measure*, and *MCC*. The performance refers to the DBSCAN algorithm.

	Confusion Matrix				Precision	Recall	Accuracy	F-Measure	*MCC*
Test Smell	**TP**	**FP**	**TN**	**FN**					
Eager Test	2,592	6,863	65	102	0.27	0.96	0.28	0.43	-0.10
Mystery Guest	1,443	7,997	98	84	0.15	0.95	0.16	0.26	-0.11
Resource Optimism	723	8,717	177	5	0.07	0.99	0.09	0.14	0.02
Test Redundancy	40	9,406	176	0	0.004	1.0	0.02	0.008	0.008

For *Test Redundancy*, both clusters have the same median values for PR and SR. As shown in Figure 4 these specific metrics contribute minimally to the detection of test smells. However, a clearer separation of clusters can be observed with Posnett_volume and Posnett_entropy for all test smells. For example, for *Eager Test*, test methods with $Posnett\_entropy < 6$ ($\overline{x} = 4.88$, and $\overline{x} = 0.2$) and $Posnett\_volume > 19$ ($\overline{x} = 489$, and $\overline{x} = 465$) are grouped into Cluster 1.

Finally, Table 7 shows the performance of the K-means and the confusion matrix for each test smell. Also, for this algorithm, the performance is low, with the *F-Measure* and *MCC* reaching a maximum of 21% and 24% respectively for *Eager Test*. In terms of *Recall*, we observed values ranging from 2% and 13%, while the maximum *Precision* reached was 78% always for *Eager Test* detection. Again, the only high metric is the *Accuracy*: the value ranges between 75% and 95%.

In summary, the performance of the two algorithms is generally low, with the K-means algorithm showing slightly better performance than the DBSCAN algorithm. To further analyze the low performance obtained, we decided to evaluate the cluster-internal quality analysis to assess their goodness. We computed *Silhouette Score* [77], *Homogeneity Score* [78], *Completeness Score* [78], *V-Measure Score* [79], and *Adjusted Rand Score* [80]. While the entire measures are reported in the online appendix [24], we noticed that all quality indexes, except for *Silhouette Score*, suggest that data were randomly assigned within the clusters. Overall, we can conclude that the cluster-based approach does not work properly in such a context.

5.3. Statistical-based Anomaly Detection

Figure 5 presents the boxplot obtained when analyzing the anomalies from a statistical point-of-view. For each test smell, we plotted the distribution of the test smell-specific metrics used in the heuristic-based approaches to detect the presence or absence of the test smells. As the reader may observe, there are several outliers for all test smells, especially for *Mystery Guest*, *Resource Optimism*, and *Test Redundancy*. In addition, for these three smells, the plot shows only the median and outliers. The data seem to be highly distributed around the median, with minimal variation within the second and third quartiles. This indicates a highly homogeneous data distribution, except for the presence of outliers. A different discussion can be drawn for *Eager Test*. For this smell, the *TMC* metric did not show any outlier, while we can observe several outliers in the *NMC* distribution. We can conclude that the data for this smell are less homogeneous compared to the other three test smells.

Table 8 reports the metrics performance obtained with this approach. Regarding performance, it is worth noting that the *F-Measure* and *MCC* reached 47% and 17%, respectively, for *Eager Test* and *Mystery Guest*, whereas for the other test smells, the metrics ranged from 1% to 31%. The *Precision* is low for all test smells(under 50%), while the *Recall* is high for all smells. Finally, the *Accuracy* ranges between 16% and 40%.

5.4. Statistical Test Results

In terms of statistics, the Friedman Test on the *F-Measure* scores of the anomaly detection methods led to the results reported in Table 9. As shown, for three of the test smells considered—*Eager Test*, *Mystery Guest*, and *Resource Optimism*—the *p*-value is lower than 0.05, confirming that there are statistically significant differences among the anomaly detection methods.

Concerning the Nemenyi Test, the results are depicted in Figure 6. We observe that three anomaly detection methods—boxplot rule, DBSCAN, and LOF—show statistically significant differences in detecting *Eager Test* and *Mystery Guest*. The blue bars with red central dots in the results indicate that these techniques have significantly different performance metrics compared to others, with the blue central dots within the blue bars further emphasizing this distinction. For *Resource Optimism* detection, only the boxplot rule and DBSCAN present significant differences. This is evident from the blue bars with red central dots, suggesting that these techniques outperform others in this specific scenario. For *Test Redundancy* detection, no significant differences are observed among the techniques. The gray bars with red central dots indicate that the performance metrics of the different techniques do not show statistically significant differences in this context.

> ☞ **Answer to RQ$_1$.** The anomaly detection approaches reached a maximum of 47% of *F-Measure* in the statistical-based approach for *Eager Test* detection, while it is between 0% and 43% for all the other approaches and test smells. In terms of *Recall*, the statistical and DBSCAN approaches achieved the highest results, ranging between 93% and 100% for the detection of all test smells, while the other approaches vary between 2% and 75%. Finally, for all smells except *Test Redundancy*, the approaches show significant differences.

5.5. Comparing anomaly detection approaches to heuristic-based and machine learning approaches

Table 10 reports the result in terms of *precision*, *recall*, *F-Measure*, and *MCC* by comparing the results coming from

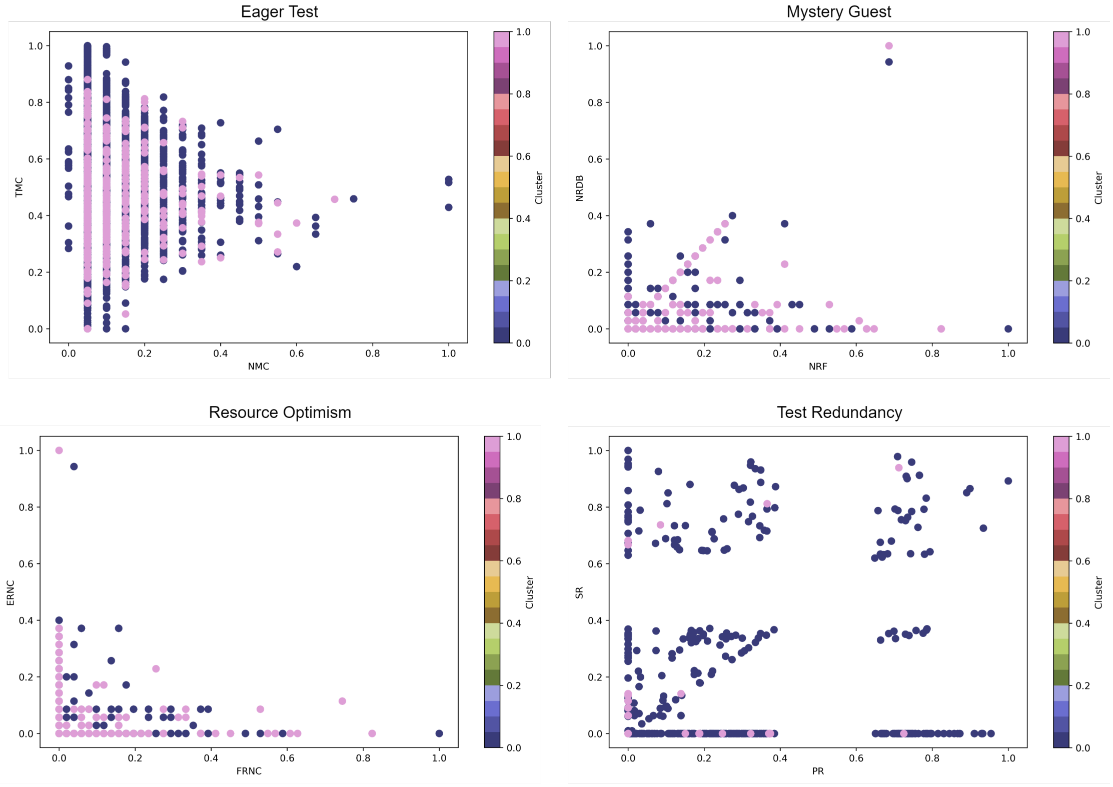

Figure 4: Scatter plot to show the spatial distribution of metrics values in clusters for all test smells using K-means.

Table 7: Results for the statistical-based approach in terms of *Precision*, *Recall*, *Accuracy*, *F-Measure*, and *MCC*. The performance refers to the K-means algorithm.

	Confusion Matrix				Precision	Recall	Accuracy	F-Measure	*MCC*
Test Smell	**TP**	**FP**	**TN**	**FN**					
Eager Test	340	92	6,836	2,354	0.78	0.13	0.75	0.21	0.24
Mystery Guest	109	323	7,772	1,418	0.25	0.07	0.82	0.11	0.05
Resource Optimism	44	388	8,506	684	0.11	0.06	0.88	0.07	0.02
Test Redundancy	1	431	9,151	39	0.002	0.02	0.95	0.004	-0.006

Table 8: Results for the statistical-based approach in terms of *Precision*, *Recall*, *Accuracy*, *F-Measure*, and *MCC*.

	Confusion Matrix				Precision	Recall	Accuracy	F-Measure	*MCC*
Test Smell	**TP**	**FP**	**TN**	**FN**					
Eager Test	2,531	5,575	1,353	163	0.31	0.93	0.40	0.47	0.16
Mystery Guest	1,527	6,804	1,291	0	0.18	1.0	0.30	0.31	0.17
Resource Optimism	728	7,563	1,331	0	0.09	1.0	0.21	0.16	0.11
Test Redundancy	40	8,094	1,488	0	0.005	1.0	0.16	0.01	0.03

Table 9: Results achieved from the application of the Friedman test.

Test Smell	*p*-value RQ$_1$	*p*-value RQ$_2$
Eager Test	$9.5902e^{-14}$	$3.1696e^{-15}$
Mystery Guest	$5.8132e^{-09}$	$8.4323e^{-09}$
Resource Optimism	$9.9534e^{-06}$	0.00049182
Test Redundancy	0.07207870	$4.8242e-06$

the anomaly detection, the heuristic- and machine learning approaches. Examining the results, heuristic-based and machine-learning approaches have performance comparable to or lower than those obtained from anomaly detection approaches. In particular, the classical machine learning methods show similar precision to the LOF algorithm, DBSCAN, and the statistical rule-based methods. For recall, machine learning demonstrates higher results for *Eager Test*, *Mystery Guest*, and *Resource Optimism* compared to the anomaly detection methods

Table 10: Results for *Precision*, *Recall*, *F-Measure*, and *MCC* for the heuristic- and ML-based approaches compared to the anomaly detection approaches.

						Precision				
	Anomaly Detection-Based ML			**Clustering**		**Statistical**	**Heuristic**			**Machine Learning**
Test Smell	IF	LOF	OCSVM	DBSCAN	K-means	Boxplot	TsDetect	DARTS	TeReDetect	
Eager Test	0.18	0.25	0.17	0.27	0.78	0.31	0.35	0.30		0.30
Mystery Guest	0.14	0.15	0.13	0.15	0.25	0.18	0.40			0.18
Resource Opt.	0.07	0.07	0.06	0.07	0.11	0.09	0.18			0.13
Test Red.	0.004	0.004	0.005	0.04	0.002	0.005			0.00	0.003

						Recall				
	Anomaly Detection-Based ML			**Clustering**		**Statistical**	**Heuristic**			**Machine Learning**
Test Smell	IF	LOF	OCSVM	DBSCAN	K-means	Boxplot	TsDetect	DARTS	TeReDetect	
Eager Test	0.43	0.54	0.28	0.96	0.12	0.93	0.16	0.31		0.60
Mystery Guest	0.60	0.56	0.38	0.94	0.07	1.0	0.40			0.75
Resource Opt.	0.61	0.56	0.40	0.99	0.06	1.0	0.37			0.62
Test Red.	0.75	0.70	0.65	1.00	0.02	1.0			0.00	0.17

						F-Measure				
	Anomaly Detection-Based ML			**Clustering**		**Statistical**	**Heuristic**			**Machine Learning**
Test Smell	IF	LOF	OCSVM	DBSCAN	K-means	Boxplot	TsDetect	DARTS	TeReDetect	
Eager Test	0.25	0.34	0.21	0.43	0.21	0.47	0.22	0.30		0.40
Mystery Guest	0.23	0.23	0.19	0.26	0.11	0.31	0.40			0.29
Resource Opt.	0.12	0.12	0.11	0.14	0.07	0.16	0.25			0.22
Test Red.	0.01	0.01	0.01	0.008	0.004	0.16			0.00	0.01

						MCC				
	Anomaly Detection-Based ML			**Clustering**		**Statistical**	**Heuristic**			**Machine Learning**
Test Smell	IF	LOF	OCSVM	DBSCAN	K-means	Boxplot	TsDetect	DARTS	TeReDetect	
Eager Test	-0.30	-0.07	-0.23	-0.09	0.24	0.16	0.06	0.03		0.07
Mystery Guest	-0.06	-0.03	-0.07	-0.11	0.05	0.17	0.29			0.10
Resource Opt.	-0.03	-0.02	-0.04	0.02	0.02	0.11	0.17			0.16
Test Red.	0.01	0.01	0.02	0.008	-0.006	0.03			-0.01	-0.004

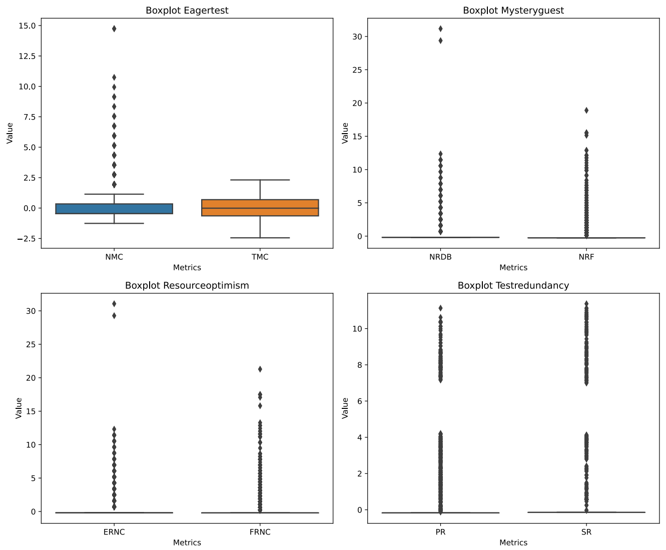

Figure 5: Boxplot representing the outliers detected for each test smell in the context of the statistical-based approach.

based on machine learning and K-means clustering. In terms of *F-Measure*, the machine learning approach outperforms some of the other methods, particularly for *Eager Test* and *Mystery Guest*, with higher *F-Measure* scores than K-means and OCSVM, and comparable results to the boxplot rule. However, in terms of MCC, the machine learning approach shows lower results than the anomaly detection-based approaches, especially for *Eager Test* detection.

As for the heuristic-based approach, we observed that anomaly detection methods perform better for *Eager Test* in terms of precision, while TsDetect achieves higher precision for *Mystery Guest* and R*esource Optimism*. In terms of *Recall*, anomaly detection methods consistently outperform the heuristic-based approaches, particularly those based on the statistical rule. Finally, in terms of *F-Measure* and *MCC*, the performance is similar, except for the detection of *Mystery Guest*, where TsDetect performs better. A different discussion can be drawn for TeReDetect, which performs worse than the anomaly detection approaches, particularly for *Recall*, where it shows a dramatic disparity (0% compared to 100%).

5.6. *Statistical Test Results*

The results of the Friedman Test are presented in Table 9. For all test smells, the *p*-value is lower than 0.05. This indicates statistically significant differences when comparing anomaly detection approaches with heuristic and machine learning approaches. To identify which approaches exhibit these statistically significant differences, we performed the Nemenyi Test. As shown in Figure 7, the DBSCAN clustering and statistical techniques consistently appear at the top of the graph for all four test smells. Additionally, for *Eager Test* and *Mystery Guest*, classical machine learning algorithms demonstrate statistically

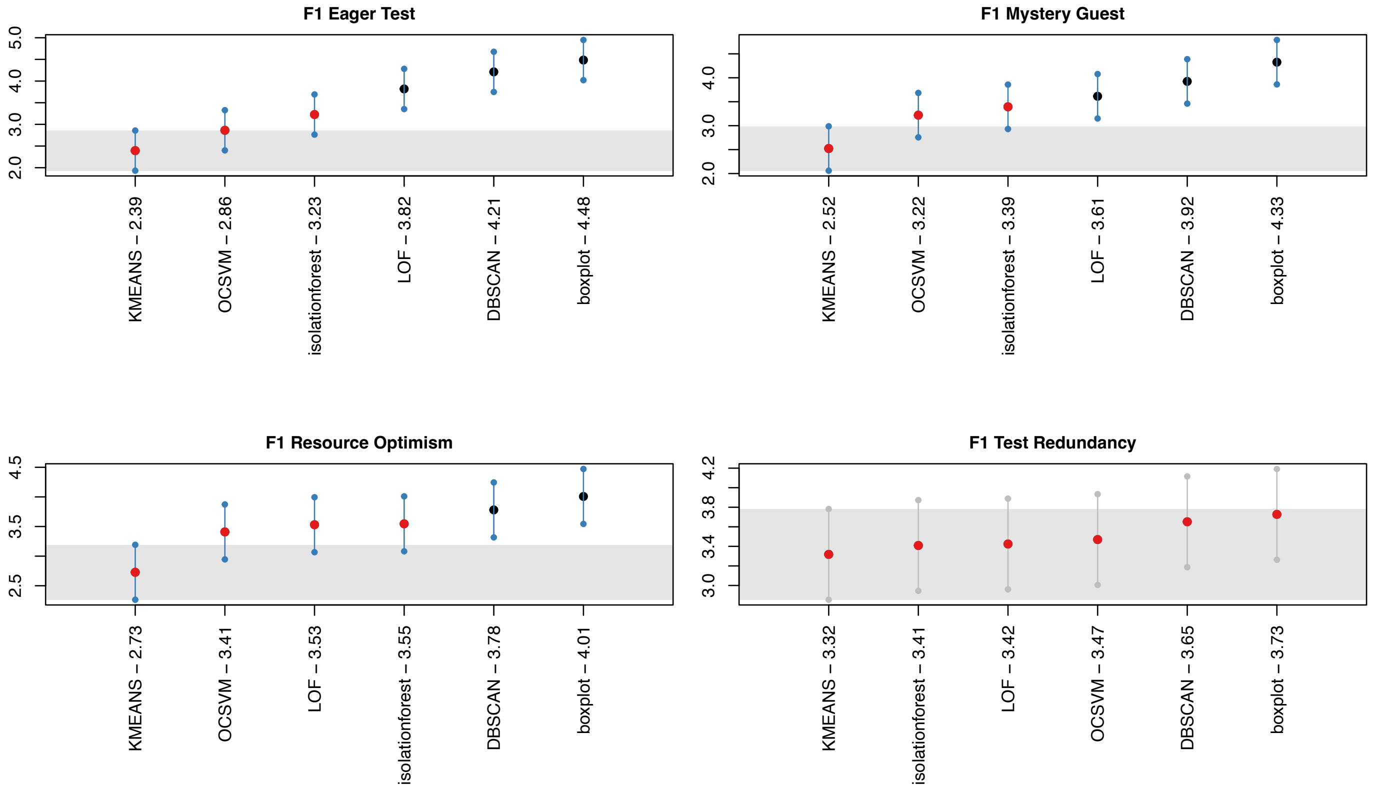

Figure 6: The likelihood of each model for the four test smells in Nemenyi rank in F-Measure. The circle dots are the median likelihood, while the error bars indicate the 95% confidence interval. 60% of likelihood means that a classification technique appears at the top rank for 60% of the studied projects.

significant differences. Lastly, heuristic-based approaches show significant differences only for *Mystery Guest*, while for the other smells, these techniques appear in the lower part of the graph or are indicated by the central red dot.

In conclusion, statistics showed that specific anomaly detection methods are more effective for certain test smells. For example, DBSCAN and statistical techniques are generally top performers across all test smells, while classical machine learning algorithms perform well in detecting *Eager Test* and *Mystery Guest*. Heuristic-based methods, although effective for *Mystery Guest*, generally rank lower for the other test smells.

☞ **Answer to RQ$_2$.** The anomaly detection approaches work better than the heuristic- and machine learning-based techniques, especially in terms of *Recall*. In terms of *F-Measure*, the values are similar when detecting *Eager Test*, *Mystery Guest*, and *Test Redundancy*, while it is slightly higher for the baselines in the detection of *Resource Optimism*. Finally, we observed statistically significant differences when analyzing the experimented techniques across all test smells.

6. Discussion and Further Analyses

Our findings revealed that the anomaly detection approaches work better in terms of *Recall*. At the same time, some additional considerations should be made to contextualize our results better and strive for further research on the matter. In particular, we first conducted an additional investigation to assess how anomaly detection and heuristic-based approaches compare when employed in a within-project scenario. On the one hand, such an analysis might reveal the impact of data sources on the performance of the experimented approaches. On the other hand, the findings obtained through this additional experimentation might help understand how the three alternative approaches might be used and complement each other in practice. In the second place, we also investigated the complementarity of the approaches in terms of the recommendations provided, in an effort to assess the extent to which their combination might potentially lead to better test smell detectors. The following sections discuss the findings obtained when performing these two additional analyses.

6.1. Anomaly Detection versus Baseline Approaches: The Impact of Data Sources

The results for **RQ$_2$** suggest that the anomaly detection approaches work better than the heuristic- and machine learning-

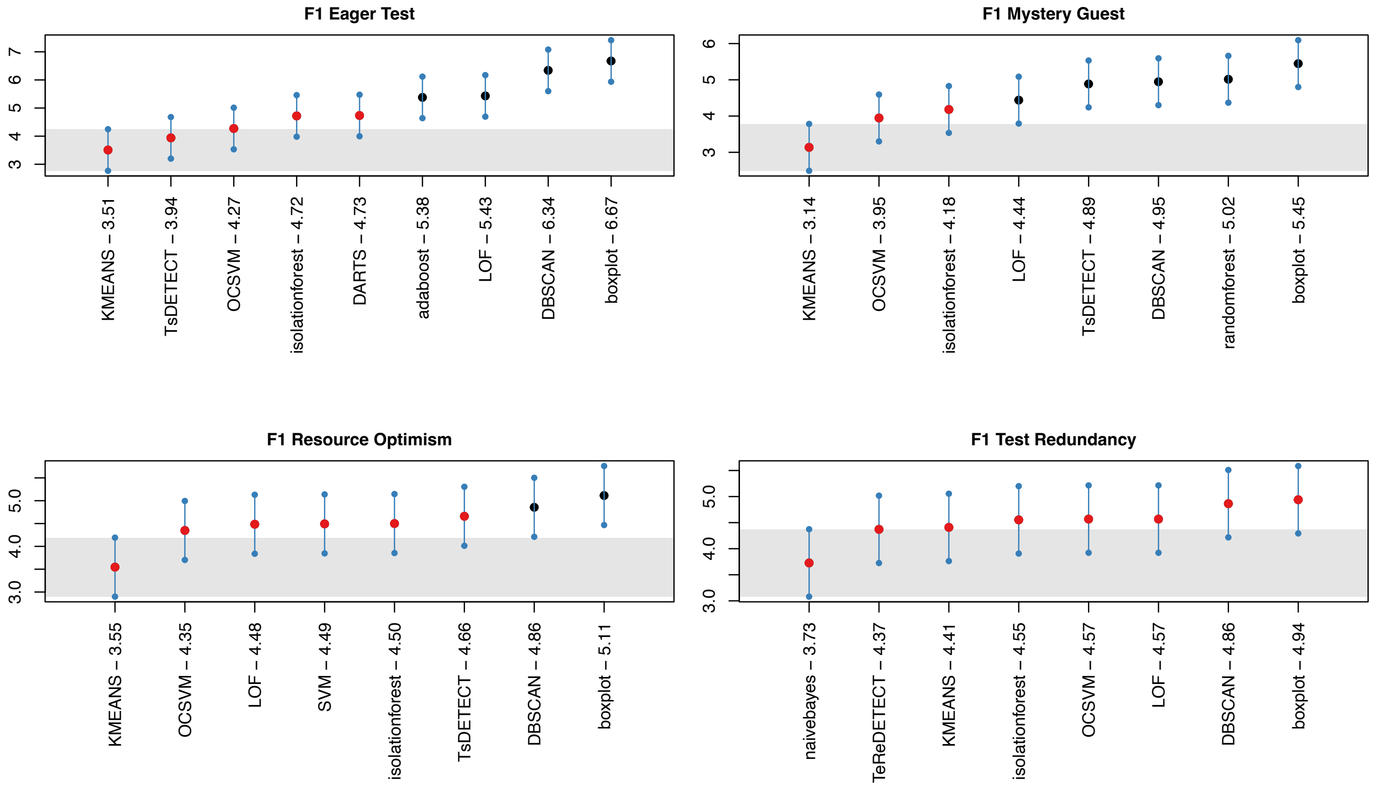

Figure 7: The likelihood of each model for the four test smells in Nemenyi rank in F-Measure. The circle dots are the median likelihood, while the error bars indicate the 95% confidence interval. 60% of likelihood means that a classification technique appears at the top rank for 60% of the studied projects.

based alternatives. On the one hand, this seems not to be enough, as the performances achieved in terms of *Precision*, *F-Measure*, and *MCC* are relatively low. On the other hand, the low performances of the baselines are particularly significant. Our findings showed that the anomaly detection methods could work better than the heuristic- and machine learning-based approaches in a real-world context, i.e., the cross-project scenario.

To further verify the generalizability of the performances observed in our study, we performed an additional analysis aiming at assessing the experimented approaches in a within-project scenario. Such an additional analysis aims to provide insights into the consistency and robustness of anomaly detection methods compared to heuristic- and machine learning-based alternatives when applied within a single project's context. In the scope of this analysis, it is worth remarking that the classical machine learning baselines were experimented with a different setting. According to the experiments performed against the same dataset as our study [21], the best machine learning models to apply in a within-project scenario are indeed different than those to apply in a cross-project scenario. To perform a fair comparison with this baseline, we, therefore, exploited these previous findings [21] and experimented with Random Forest with Borderline-SMOTE for detecting *Eager Test* and *Resource Optimism*, Random Forest with no balancing for *Mystery Guest* detection, and Naive Bayes with no balancing for *Test Redundancy* detection. Table 11 reports the results achieved. From the table, we can observe that the performances exhibit several notable differences. While in **RQ**$_2$ the anomaly detection-based approaches generally performed better, in the within-project scenario the experimented baselines achieved higher performances in some metrics.

Specifically, looking at the *Precision* and *MCC*, the machine learning approach achieved the highest precision for *Eager Test* (76%), *Mystery Guest* (76%), and *Resource Optimism* (58%). For *F-Measure*, the classical machine learning algorithms also showed better performance for *Eager Test* with scores of 75%, significantly higher than all the other techniques, which range between 23% and 45%. A similar discussion can be drawn for the detection of *Mystery Guest* and *Resource Optimism*: the results show that the classical machine learning algorithms outperformed the other approaches.

In terms of *Recall*, the DBSCAN algorithm reached the highest performance for all test smells, suggesting that the algorithm detects almost all true positives. Despite this, the low *Precision* for this anomaly detection approach suggests that the algorithm may include many irrelevant instances.

Finally, analyzing *Test Redundancy*, we can observe that almost all performance indicators are low or similar to the cross-

Table 11: Results for *Precision*, *Recall*, *F-Measure*, and *MCC* for all the approaches in a within-project scenario.

						Precision				
	Anomaly Detection-Based ML			**Clustering**		**Statistical**		**Heuristic**		**Machine Learning**
Test Smell	IF	LOF	OCSVM	DBSCAN	K-means	Boxplot	TsDetect	DARTS	TeReDetect	
Eager Test	0.23	0.25	0.22	0.27	0.23	0.31	0.37	0.37		0.76
Mystery Guest	0.16	0.17	0.16	0.15	0.16	0.18	0.42			0.76
Resource Opt.	0.10	0.10	0.09	0.08	0.11	0.09	0.21			0.58
Test Red.	0.008	0.007	0.01	0.004	0.001	0.005			0.00	0.01

						Recall				
	Anomaly Detection-Based ML			**Clustering**		**Statistical**		**Heuristic**		**Machine Learning**
Test Smell	IF	LOF	OCSVM	DBSCAN	K-means	Boxplot	TsDetect	DARTS	TeReDetect	
Eager Test	0.55	0.61	0.36	0.97	0.18	0.83	0.17	0.31		0.75
Mystery Guest	0.62	0.68	0.43	0.95	0.23	0.87	0.44			0.49
Resource Opt.	0.63	0.66	0.41	1.0	0.33	0.87	0.37			0.39
Test Red.	0.73	0.65	0.68	1.0	0.05	1.0			0.00	0.43

						F-Measure				
	Anomaly Detection-Based ML			**Clustering**		**Statistical**		**Heuristic**		**Machine Learning**
Test Smell	IF	LOF	OCSVM	DBSCAN	K-means	Boxplot	TsDetect	DARTS	TeReDetect	
Eager Test	0.32	0.36	0.27	0.43	0.20	0.45	0.23	0.32		0.75
Mystery Guest	0.26	0.28	0.26	0.20	0.11	0.30	0.44			0.60
Resource Opt.	0.17	0.18	0.15	0.14	0.17	0.16	0.27			0.46
Test Red.	0.01	0.01	0.02	0.008	0.001	0.01			0.00	0.02

						MCC				
	Anomaly Detection-Based ML			**Clustering**		**Statistical**		**Heuristic**		**Machine Learning**
Test Smell	IF	LOF	OCSVM	DBSCAN	K-means	Boxplot	TsDetect	DARTS	TeReDetect	
Eager Test	-0.8	-0.11	-0.15	-0.1	-0.06	0.11	0.06	0.04		0.66
Mystery Guest	-0.06	-0.02	-0.04	-0.12	0.01	0.11	0.29			0.55
Resource Opt.	-0.03	-0.02	-0.04	0.03	0.07	0.08	0.15			0.41
Test Red.	0.01	-0.01	0.03	0.007	-0.03	0.04			-0.01	0.04

project scenario. The low overall performance suggests that the test cases deemed anomalies are not sufficiently different, leading to suboptimal detection by anomaly detection approaches.

In conclusion of this analysis, while anomaly detection methods showed promised results in cross-project scenarios, their performance metrics were not consistently high. In within-project scenarios, classical machine learning algorithms outperformed anomaly detection in specific metrics, possibly suggesting the need for novel approaches that may support the selection of the best test smell detector to use on the basis of the context considered and the data sources available. As a side note, the higher performances observed for classical machine learning with respect to what was reported in literature [21] calls for a reflection. Our experimental settings differed from previous studies: our machine learning baselines were indeed trained using not only the metrics considered by previous heuristic-based test smell detectors [14, 15, 48] but also additional test code metrics related to size, complexity, and readability. These enhanced performances indicate that incorporating metrics that characterize test code quality from various perspectives can improve the detection accuracy of machine learning models. In this sense, our work provides a side contribution to the research community, paving the way of further refinement of machine learning approaches for test smell detection.

Additional conclusions can be drawn by qualitatively analyzing the results of the anomaly detection methods in the within-project scenario. More specifically, we analyzed the potential reasons why these methods have lower performance than the baselines, finding the limited data variability as the main reason. Specifically, anomaly detection methods rely on identifying patterns that deviate from the norm. In a within-project scenario, the variability of test cases might be limited because the coding practices and test structures are relatively consistent within the same project. This limited variability makes it challenging for anomaly detection methods to distinguish between normal and anomalous behavior effectively. To verify this conjecture, we conducted additional analyses. Specifically, we computed multiple structural and semantic metrics to establish the similarity between the test classes of the considered projects. For each project, we first extracted the set of test classes and methods available. From a structural perspective, we computed (1) the number of distinct developers involved in the development of the test classes; (2) the number of test methods per class; (3) the average method length; (4) the number of assertions per test method; (5) the use of `setup` and `teardown` methods; (6) the distribution of test methods across different packages; and (7) the frequency of specific testing frameworks or libraries used. From a semantic perspective, we computed (1) the lexical similarity between test method names, using the cosine similarity metric [81]; and (2) the similarity in the structure of test cases, considering the adherence of the tests to the arrangement-act-assert pattern; (3) the reuse of test utility methods across different test classes; (4) the use of comments and documentation within test methods. Altogether, these metrics can effectively assess the similarity of test classes within individual projects by capturing both the structural consistency and the semantic uniformity of the tests, which reflect the standardized practices and conventions used throughout the project.

Upon collection of these metrics, we analyzed the distribution of the various metrics by computing boxplots and assessing the variability of the metric values across the test classes of individual projects. Our findings confirm the limited variability of data. Indeed, for all the considered projects, the distribution of all metric values was tightly clustered around their respective medians, with small interquartile ranges and few outliers. This suggests that the test classes within each project exhibit high consistency in both structural and semantic characteristics, reinforcing our argument that the limited variability challenges the effectiveness of anomaly detection methods in distinguishing between normal and anomalous behaviors.

As part of our online appendix [24], we included a technical report explaining how we computed the metrics considered in such an additional analysis as well as the detailed results.

6.2. Anomaly Detection versus Baseline Approaches: The Qualitative Perspective

To provide a broader overview of the low performance observed, we performed an additional qualitative analysis of the false positives and negatives output by the experimented approaches, seeking possible reasons for failure. In particular, the two first authors of the paper have jointly and systematically analyzed the set of false positive and negative instances for each approach. Each instance was reviewed in detail, focusing on (1) the context and nature of the test smells considered, and (2) the consistency between the metrics computed and the actual content of the test methods: in this way, the two inspectors could identify the elements of the test cases that were not properly measured by metrics or that biased the metric computation. The inspection process lasted 50 person-hour. At the end, the other authors of the paper were involved. The two original inspectors presented the results of the qualitative analysis, guiding a discussion on the main findings achieved. Such a discussion allowed us to refine some considerations and led to the final set of conclusions drawn, which are discussed below.

Starting from *Eager Test*, traditional heuristic-based tools implement detection rules that essentially rely on the number of method calls made by a test method, identifying a smell if the test method makes more than one call. Alternative metrics, like the textual ones employed by DARTS, gather similar pieces of information, extracting the textual similarity between the methods called by a test case. While these detection rules are in line with the original definition of the test smell, they do not fully account for the complexity of unit testing. There are two specific issues that should be remarked:

- Test methods may perform auxiliary method calls to set the testing environment. These auxiliary calls might include additional initialization steps than those performed by the setup method: for instance, initializing mock objects, setting up database connections, and configuring external services. Such auxiliary calls are not directly related to the core functionality being tested but are necessary for setting the test environment. As result, these methods can lead to false positives when detecting *Eager Test* smells since the detection rule does not differentiate between auxiliary and actual method calls intended to test specific functionalities of the production code. As an exemplary case, consider the test reported in Listing 1, which belongs to the wro4j project available in the dataset. The test is not smelly, yet all the heuristic approaches identified it as such.

```
  @Test
  public void testDoFilterInDEPLOYMENTMode()
    throws Exception {
  when(mockRequest.getRequestURI()).
    thenReturn("/g2.js");
  victim.setWroManagerFactory(
    createValidManagerFactory());
setConfigurationMode(
    FilterConfigWroConfigurationFactory.
    PARAM_VALUE_DEPLOYMENT);
  victim.init(mockFilterConfig);
  victim.doFilter(mockRequest, mockResponse,
    mockFilterChain);
  }
```

Listing 1: Example of false positive *Eager Test* due to mock objects.

  In similar cases, the anomaly detectors could sometimes properly label the smelliness of the test. For example, the detector based on the analysis of the statistical boxplot could not identify an outlier with respect to the whole distribution of the NMC metric, hence correctly labeling the test as non-smelly. On the contrary, the machine learning-based anomaly detectors failed in most of the cases, being biased by the use of auxiliary methods.

- Existing tools do not explicitly consider the difference between intra-method and intra-class unit testing. Intra-method unit testing involves testing individual methods within the same class, while intra-class unit testing involves testing interactions between methods within the same class. As such, the detection rules may wrongly flag smelly test cases when tests perform intra-class unit testing. For example, a test method that calls multiple helper methods within the same class to verify the overall behavior might be incorrectly identified as an *Eager Test* smell, even though it is testing a coherent and focused functionality. An example of an intra-class test comes from the JFreeChart project and it is shown below in Listing 2.

```
@Test
  public void testPrune() {
      DefaultTableXYDataset dataset = new
    DefaultTableXYDataset();
      dataset.addSeries(createSeries1());
      dataset.removeSeries(1);
      dataset.prune();
      assertEquals(4,dataset.getItemCount());
  }
```

Listing 2: Example of false positive *Eager Test*.

  According to our qualitative investigation, this issue biased all the detection approaches experimented, hence representing a critical concern for test smell detection.

A similar discussion can be drawn for the other test smells considered in the study. As for *Resource Optimism*, which involves tests relying on external resources without verifying their status, current metrics are not specifically defined to account for multiple conditions that may arise in unit testing. In this case, a representative example is the use of mocks, i.e., simulated objects that mimic the behavior of real objects in controlled ways: mocked objects should not be considered as part of the computation of the metrics, but they instead do. This miscalculation can lead to incorrect identification of *Resource Optimism* smells, as the presence of mocks indicates a controlled testing environment rather than reliance on actual external resources. An example of a test case that was wrongly classified by all the approaches is shown in Listing 3:

```
@Test
  public void testAggregatedComputedFolder2()
    throws Exception {
  final HttpServletRequest request = Mockito.mock
    (HttpServletRequest.class);
  final HttpServletResponse response = Context.
    get().getResponse();
  Mockito.when(request.getRequestURI()).
    thenReturn("/wro4j/wro/path/to/g1.css");
  Context.unset();
  Context.set(Context.webContext(request,
    response, Mockito.mock(FilterConfig.class)));

  managerFactory.create().process();

  Assert.assertEquals("/wro4j/wro/path/to/",
    Context.get().getAggregatedFolderPath());
  }
```

Listing 3: Example of false positive *Resource Optimism* due to mock object.

We found that mocks systematically influence the detection performance, representing a key concern that may bias the correct interpretation of the smelliness of test code.

On a similar note, *Mystery Guest*, which refers to tests using external resources without clear visibility or initialization within the test itself, can also be misidentified. For example, if a test method uses configuration files or environment variables that are not explicitly initialized within the test, current metrics might incorrectly flag these tests as not having a *Mystery Guest* smell, even though the use of external resources might be well-documented and properly managed elsewhere in the test suite. For instance, consider the example reported in Listing 4, coming from the dropwizard project. Specifically, the `setup` method prepares a servlet environment with several servlets registered to handle specific URL patterns, start the server, and configure MIME types for certain file extensions. Looking at the test case, the code adds an additional resource, i.e., `foo.mp4`, before verifying that the server correctly supports byte-range requests for media files.

```
 @BeforeClass
  public static void startServletTester() throws
    Exception {
      SERVLET_TESTER.addServlet(DummyAssetServlet
    .class, DUMMY_SERVLET + ’*’);
      SERVLET_TESTER.addServlet(
    NoIndexAssetServlet.class, NOINDEX_SERVLET +
    ’*’);
      SERVLET_TESTER.addServlet(
    NoCharsetAssetServlet.class,
    NOCHARSET_SERVLET + ’*’);
      SERVLET_TESTER.addServlet(RootAssetServlet.
    class, ROOT_SERVLET + ’*’);
      SERVLET_TESTER.start();
    SERVLET_TESTER.getContext().getMimeTypes().
    addMimeMapping("mp4", "video/mp4");
    SERVLET_TESTER.getContext().getMimeTypes().
    addMimeMapping("m4a", "audio/mp4");
  }
```

```
 @Test
  public void supportsByteRangeForMedia() throws
    Exception {
      request.setURI(ROOT_SERVLET + "assets/foo.
    mp4");
      response = HttpTester.parseResponse(
    SERVLET_TESTER.getResponses(request.generate
    ()));
      assertThat(response.getStatus()).isEqualTo
    (200);    assertThat(response.get(HttpHeader.
    ACCEPT_RANGES)).isEqualTo("bytes");
      request.setURI(ROOT_SERVLET + "assets/foo.
    m4a");
      response = HttpTester.parseResponse(
    SERVLET_TESTER.getResponses(request.generate
    ()));
      assertThat(response.getStatus()).isEqualTo
    (200);    assertThat(response.get(HttpHeader.
    ACCEPT_RANGES)).isEqualTo("bytes");
  }
```

Listing 4: Example of false negative *Mystery Guest*.

Our findings suggest that (1) current detectors may not properly handle this situation; and (2) a contextual gathering phase focused on the preliminary identification of shared or global setup procedures could help differentiate between legitimate use of external resources and actual *Mystery Guest* smells.

Finally, as for *Test Redundancy*, which occurs when multiple tests cover the same functionality unnecessarily, current metrics and tools rely on code coverage information. Despite being a valuable source, coverage may not provide a comprehensive view on the matter. Two tests may cover a similar path in production code yet not be redundant if they verify different aspects of the functionality. For example, they may verify two special inputs that lead to failures. For instance, consider the two tests reported in Listing 5. They both cover a similar path, yet they exercise the production code for two different edge cases that may lead to defective behaviors.

```
@Test
  public void testGlobFilter() throws Exception {
  createHttpFSServer(false, false);
  String user = HadoopUsersConfTestHelper.
    getHadoopUsers()[0];
  URL url = new URL(TestJettyHelper.getJettyURL()
    ,
      MessageFormat.format(
          "/webhdfs/v1/tmp?user.name={0}&op=
    liststatus&filter=f*", user));
  HttpURLConnection conn = (HttpURLConnection)
    url.openConnection();
  Assert.assertEquals(conn.getResponseCode(),
    HttpURLConnection.HTTP_OK);
  BufferedReader reader = new BufferedReader(
      new InputStreamReader(conn.getInputStream()
    ));
  reader.readLine();
  reader.close();
  }


@Test
  public void testHdfsAccess() throws Exception {
  createHttpFSServer(false, false);

  String user = HadoopUsersConfTestHelper.
    getHadoopUsers()[0];
  URL url = new URL(TestJettyHelper.getJettyURL()
    ,
      MessageFormat.format("/webhdfs/v1/?user.
    name={0}&op=liststatus",
                                  user));
  HttpURLConnection conn = (HttpURLConnection)
    url.openConnection();
  Assert.assertEquals(conn.getResponseCode(),
    HttpURLConnection.HTTP_OK);
  BufferedReader reader = new BufferedReader(
      new InputStreamReader(conn.getInputStream()
    ));
  reader.readLine();
  reader.close();
  }
```

Listing 5: Example of false positive *Test Redundancy*.

Also in this case, we found that this aspect biases all the experimented approaches. As a consequence, additional sources of information, like semantic or textual information, should be assessed. Especially in the era of large language models, integrating such information could enhance the detection of truly redundant tests. Because of the reasons above, we argue that (1) novel metrics should account for the complexity of unit testing and potentially highlight contextual factors that may influence the evaluation of the smelliness of test artifacts; (2) complementary pre-processing activities would be beneficial to guide test smell detection. For instance, test smell detection tools could be accompanied by a preliminary classification step that helps discriminate between intra-method and intra-class test cases, reducing the risk of misclassification of smelly instances. At the same time, we also highlight the need for the research community to revisit well-established definitions that have remained unchanged for decades. For instance, the original definition of *Eager Test* indeed provides a generic statement that might be worth further elaboration in order to encourage researchers to develop more detailed and nuanced definitions.

Upon the collection of the performance of the heuristic-based approaches, we also inquired about the potential reasons behind the different performance observed when executing the heuristic-based approaches with respect to the original evaluations of these tools. In this respect, while we cannot speculate on the reasons behind the differences observed, we can provide two major observations:

- Most previous works have been assessed on a smaller scale. For example, TsDetect was evaluated on a total of 65 annotated test cases, while DARTS was assessed on a set of 494 test smell instances. In contrast, our study provides greater ecological validity by evaluating these approaches on 5,003 individual smell instances. The larger experimental size in our study may contribute to the observed differences in performance.
- Regarding TeReDetect, the original manuscript evaluated the approach using the mutation analysis technique to measure the fault detection effectiveness of the reduced test suite of a real Java project. This type of experimentation was different, focusing on the impact of detection on mutation analysis rather than on assessing detection accuracy directly. Consequently, our work offers the first assessment specifically targeting the detection accuracy of this tool, which may explain the discrepancies.

These observations suggest that the differences in performance could be due to the larger scale of our dataset and the different experimental focuses compared to previous studies.

6.3. Anomaly Detection versus Baseline Approaches: The Role of Complementarity

Our study reveals that the anomaly detection approaches could work in the context of test smells detection. Nonetheless, we need to understand whether some approaches analyzed may work better than others and whether there are complementarities among them.

To further elaborate on the differences between the approaches analyzed, we conducted an additional analysis among the experimented approaches. Given two approaches m_i and m_j, we computed (1) the amount of test smells correctly predicted by both m_i and m_j ($m_i \bigcap m_j$), (2) the amount of test smells correctly predicted by m_i only and missed by m_j ($m_i \backslash m_j$), and (3) the amount of test smells correctly predicted by m_j only and missed by m_i ($m_j \backslash m_i$). In addition, given the four experimented approaches, m_i, m_j, m_k, and m_p, we computed (1) the amount of test smells correctly predicted by m_i only and missed by mj, m_k, and m_p ($m_i \backslash (mj \bigcup mk \bigcup mp)$), and (2) the amount of test smells correctly predicted by all approaches (($mi \bigcap mj \bigcap mk \bigcap mp) \backslash (m_i \bigcup mj \bigcup mk \bigcup mp$)).

Table 12 presents the results achieved when assessing the overlap, and Table 13 presents the results for the complementarity analyses. When comparing each approach, we could observe that most of the correct predictions are common in the statistical and cluster-based approaches, i.e., 90.16% for *Eager Test*, 94.49% for *Mystery Guest*, 93.31% for *Resource Optimism*, and 100% for *Test Redundancy*. The correct prediction between the other approaches is different for each test smell. It implies that the detection of test smells may vary depending on the approach employed. In addition, the machine learning-based anomaly detection methods complement the heuristic-based approaches for detecting all test smells, 23.85% for *Eager Test*, and 100% for the other test smells.

Considering the complementary results combining the anomaly detection approaches, we can observe that the heuristic and ML-based approaches do not complement the combination of other approaches. However, statistical approaches complement the others in 1.74% for *Eager Test*, and 0.06% for *Mystery Guest*. While the contribution of statistical approaches is

Table 12: Overlap results for the comparison of the anomaly detection and heuristic-based approaches.

Eager Test			Mystery Guest			Resource Optimism			Test Redundancy		
Stats ⋂ Cluster	Stats \ Cluster	Cluster \ Stats	Stats ⋂ Cluster	Stats \ Cluster	Cluster \ Stats	Stats ⋂ Cluster	Stats \ Cluster	Cluster \ Stats	Stats ⋂ Cluster	Stats \ Cluster	Cluster \ Stats
90.16%	3.78%	6.05%	94.49%	5.50%	0.00%	93.31%	0.68%	0.00%	100.00%	0.00%	0.00%
Stats ⋂ Heur.	Stats \ Heur.	Heur. \ Stats	Stats ⋂ Heur.	Stats \ Heur.	Heur. \ Stats	Stats ⋂ Heur.	Stats \ Heur.	Heur. \ Stats	Stats ⋂ Heur.	Stats \ Heur.	Heur. \ Stats
37.18%	60.64%	2.16%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%
Stats ⋂ ML	Stats \ ML	ML \ Stats	Stats ⋂ ML	Stats \ ML	ML \ Stats	Stats ⋂ ML	Stats \ ML	ML \ Stats	Stats ⋂ ML	Stats \ ML	ML \ Stats
18.68%	80.65%	0.66%	27.30%	72.69%	0.0%	38.04%	61.95%	0.0%	32.65%	67.5%	0.0%
Stats ⋂ ML-C	Stats \ ML-C	ML-C \ Stats	Stats ⋂ ML-C	Stats \ ML-C	ML-C \ Stats	Stats ⋂ ML-C	Stats \ ML-C	ML-C \ Stats	Stats ⋂ ML-C	Stats \ ML-C	ML-C \ Stats
54.82%	41.73%	3.43%	67.25%	32.74%	0.0%	36.26%	63.73%	0.0%	22.50%	77.50%	0.0%
Cluster ⋂ Heur.	Cluster \ Heur.	Heur. \ Cluster	Cluster ⋂ Heur.	Cluster \ Heur.	Heur. \ Cluster	Cluster ⋂ Heur.	Cluster \ Heur.	Heur. \ Cluster	Cluster ⋂ Heur.	Cluster \ Heur.	Heur. \ Cluster
37.47%	61.23%	1.29%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%
Cluster ⋂ ML	Cluster \ ML	ML \ Cluster	Cluster ⋂ ML	Cluster \ ML	ML \ Cluster	Cluster ⋂ ML	Cluster \ ML	ML \ Cluster	Cluster ⋂ ML	Cluster \ ML	ML \ Cluster
18.79%	81.01%	0.19%	28.18%	71.26%	0.55%	37.74%	61.84%	0.41%	32.50%	67.50%	0.00%
Cluster ⋂ ML-C	Cluster \ ML-C	ML-C \ Cluster	Cluster ⋂ ML-C	Cluster \ ML-C	ML-C \ Cluster	Cluster ⋂ ML-C	Cluster \ ML-C	ML-C \ Cluster	Cluster ⋂ ML-C	Cluster \ ML-C	ML-C \ Cluster
56.97%	41.80%	1.21%	61.86%	32.69%	5.43%	35.76%	63.68%	0.55%	22.50%	77.50%	0.00%
Heur. ⋂ ML	Heur. \ ML	ML \ Heur.	Heur. ⋂ ML	Heur. \ ML	ML \ Heur.	Heur. ⋂ ML	Heur. \ ML	ML \ Heur.	Heur. ⋂ ML	Heur. \ ML	ML \ Heur.
13.01%	63.12%	23.85%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%
Heur. ⋂ ML-C	Heur. \ ML-C	ML-C \ Heur.	Heur. ⋂ ML-C	Heur. \ ML-C	ML-C \ Heur.	Heur. ⋂ ML-C	Heur. \ ML-C	ML-C \ Heur.	Heur. ⋂ ML-C	Heur. \ ML-C	ML-C \ Heur.
28.27%	23.03%	48.68%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%
ML ⋂ ML-C	ML \ ML-C	ML-C \ ML	ML ⋂ ML-C	ML \ ML-C	ML-C \ ML	ML ⋂ ML-C	ML \ ML-C	ML-C \ ML	ML ⋂ ML-C	ML \ ML-C	ML-C \ ML
16.15%	12.19%	71.65%	23.52%	12.14%	64.32%	16.09%	43.34%	40.55%	22.22%	50.00%	27.88%

Table 13: Complementarity results for combining the anomaly detection and heuristic-based approaches.

Eager Test		Mystery Guest	
Stats \ (Cluster ⋃ Heur. ⋃ ML ⋃ ML-C)	Heur. \ (Stats ⋃ Cluster ⋃ ML ⋃ ML-C)	Stats \ (Cluster ⋃ Heur. ⋃ ML ⋃ ML-C)	Heur. \ (Stats ⋃ Cluster ⋃ ML ⋃ ML-C)
1.74%	0.00%	0.06%	0.00%
Cluster \ (Stats ⋃ Heur. ⋃ ML ⋃ ML-C)	ML \ (Stats ⋃ Cluster ⋃ Heur. ⋃ ML-C)	Cluster \ (Stats ⋃ Heur. ⋃ ML ⋃ ML-C)	ML \ (Stats ⋃ Cluster ⋃ Heur. ⋃ ML-C)
1.63%	0.00%	0.00%	0.00%
ML-C \ (Heur. ⋃ ML ⋃ Stats ⋃ Cluster)		(ML-C \ (Heur. ⋃ ML ⋃ Stats ⋃ Cluster)	
0.00%		0.00%	
(Stats ⋂ Cluster ⋂ Heur. ⋂ ML) \ (Stats ⋃ Cluster ⋃ Heur. ⋃ ML)		(Stats ⋂ Cluster ⋂ Heur. ⋂ ML) \ (Stats ⋃ Cluster ⋃ Heur. ⋃ ML)	
0.00%		0.00%	
Resource Optimism		**Test Redundancy**	
Stats \ (Cluster ⋃ Heur. ⋃ ML ⋃ ML-C)	Heur. \ (Stats ⋃ Cluster ⋃ ML ⋃ ML-C)	Stats \ (Cluster ⋃ Heur. ⋃ ML ⋃ ML-C)	Heur. \ (Stats ⋃ Cluster ⋃ ML ⋃ ML-C)
0.0%	0.00%	0.00%	0.00%
Cluster \ (Stats ⋃ Heur. ⋃ ML ⋃ ML-C)	ML \ (Stats ⋃ Cluster ⋃ Heur. ⋃ ML-C)	Cluster \ (Stats ⋃ Heur. ⋃ ML ⋃ ML-C)	ML \ (Stats ⋃ Cluster ⋃ Heur. ⋃ ML-C)
0.00%	0.00%	0.00%	0.00%
ML-C \ (Heur. ⋃ ML ⋃ Stats ⋃ Cluster)		(ML-C \ (Heur. ⋃ ML ⋃ Stats ⋃ Cluster)	
0.00%		0.00%	
(Stats ⋂ Cluster ⋂ Heur. ⋂ ML) \ (Stats ⋃ Cluster ⋃ Heur. ⋃ ML)		(Stats ⋂ Cluster ⋂ Heur. ⋂ ML) \ (Stats ⋃ Cluster ⋃ Heur. ⋃ ML)	
0.00%		0.00%	

not significant, their positive impact on most test smells suggests they could be a useful component in a comprehensive anomaly detection strategy.

Based on the previously discussed observations, the approaches appear to be complementary. This synergy suggests the potential for developing a meta-approach that dynamically selects the most suitable technique for detecting test smells, depending on the specific characteristics of the test case. Such a meta-approach would leverage the strengths of each individual technique, optimizing test smell detection by applying the most effective method for each unique scenario.

7. Threats To Validity

This section discusses the potential threats that may affect the validity of our empirical study.

Construct validity. The first concern revolves around the test smells dataset exploited in our research. We based our investigation on a large-scale and curated dataset released in previous work [21], which comprises 3,652 smelly test cases out of 9,633 test cases. However, it is important to acknowledge that the dataset might not entirely capture the broad perspective of developers on test smells due to their intrinsically subjective nature. A second threat concerns the computation of independent variables, which relied on metrics previously defined in the literature. We specifically collected and utilized metrics established in test smell research to ensure a fair comparison between heuristic-based approaches and anomaly detection methods. This methodological choice allows for a direct and equitable evaluation of the different detection techniques. As part of our future research agenda, we plan to identify the limitations of the current metrics and develop new features to enhance test smell detection. Furthermore, our study incorporated an additional set of metrics able to characterize test code quality from various perspectives, potentially boosting the overall detection capabilities of anomaly detection methods. Concurrently, we evaluated the role of these additional metrics in improving existing test smell detectors, including machine learning-based approaches, thereby exploring avenues for refining and advancing test smell detection methodologies.

External Validity. As for the generalizability of the conclusions, the main threat concerns the subject projects of our study. The dataset exploited was based on 66 Java open-source projects from GitHub, which are only a fraction of the complete picture of open-source software. Therefore, we cannot ensure that our findings generalize when considering different programming languages or other software systems. In this regard, we made all materials and scripts as part of an online appendix to stimulate further research that may corroborate our findings in different contexts [24].

In the second place, we had to limit the analysis to Java for a key reason. Most of the tools used in the study are only available for Java [17, 47]: this is true for all the heuristic approaches exploited, but also for the additional data analytics instruments employed to collect metrics, e.g., the analytical tool used to compute readability metrics [50]. To the best of our knowledge, current research does not provide similar data analytics tools and heuristic test smell detectors that may be used to run our study on projects written in different programming languages. Consequently, expanding the study to include diverse programming languages was unfeasible, as it would have required building from scratch - and, more importantly, validating - the tools required to conduct the study.

While this threat cannot be fully mitigated, two key points need to be discussed. First, according to the 2023 report from JetBrains,[2] Java remains one of the most popular programming languages, with 49% of developers still using it. This suggests that our findings remain highly relevant to a significant portion of the software development community. Given the Java's widespread, especially in large-scale, enterprise-level systems, our results offer valuable insights into a substantial segment of the open-source ecosystem. Then, while our study is focused on Java, the performance issues we identified are rooted in the process of test smell detection rather than the specific programming language. Test smells, such as *Mystery Guest*, are defined in ways that are not inherently related to any single language. Therefore, the qualitative insights we gained from this study are likely to generalize to other object-oriented languages. While more studies in different languages are needed, we believe our findings are broadly applicable because they address fundamental aspects of test smell definitions and detection.

Another threat concerns the generalizability of industry-scale projects. We acknowledge that there are differences between open-source and industrial projects, particularly in terms of development processes and testing strategies, which may influence the presence of test smells. However, it is important to emphasize that we performed additional analysis on the characteristics of the projects to evaluate the structural and semantic profiles, as reported in Section 6.1. These metrics offer a solid foundation for assessing the applicability of our findings to industry-scale projects with similar characteristics. For instance, the distribution of test methods across packages and the reuse of test utility methods are practices that scale with project size, whether open-source or proprietary. Similarly, the use of testing frameworks and the presence of setup and teardown methods are fundamental aspects of test quality assurance, regardless of the project's scope or environment.

Conclusion validity. This category refers to the relation between treatment and outcome. We did not have a baseline for the anomaly detection methods experimented with, as our work represents the first attempt to study these approaches for test smell detection. As such, we experimented with multiple techniques to understand their performance. Our online appendix [24] includes all our findings, which researchers can use to understand further the impact of anomaly detection techniques for test smell detection. We acknowledge the inherent challenges in comparing different approaches due to variations in how test smells are detected and how performance is evaluated. For instance, heuristic-based methods rely on specific thresholds, while anomaly detection techniques identify outliers based on data characteristics, making direct comparisons difficult. Additionally, some methods may perform better in different project contexts, as noted in Section 6.1. To strengthen our comparison and the conclusions drawn in the study, we applied statistical analyses to verify the differences observed between (1) the experimented anomaly detection methods in $\mathbf{RQ}_1$ and (2) the anomaly detection methods and heuristic- and machine learning-based baselines in $\mathbf{RQ}_2$. Specifically, we first verified the normality of our data using the Shapiro-Wilk test [76]. Then, we computed appropriate statistical tests, such as the Friedman and Nemenyi tests [74, 75], to determine if there were any statistically significant differences in terms of the *F-Measure* scores achieved by the experimented approaches.

Internal Validity. This category of threat concerns the selection and implementation of anomaly detection-based approaches to conduct our empirical study on test smell detection. To address this, we employed robust and well-known anomaly detection approaches that are widely used in such a context [2].

8. Conclusion and Future Work

In this paper, we performed an empirical study to assess the performance of anomaly detection approaches to detect test smells. We considered a dataset with a total of 9,633 test cases from 66 open-source Java projects, with 2,699 instances of *Eager Test*, 1,534 instances of *Mystery Guest*, 730 instances of *Resource Optimism*, and 40 instances of *Test Redundancy*.

First, in $\mathbf{RQ}_1$, we investigated how anomaly detection detects test smells. As a result, their performance in detecting test smells did not exceed 47% for *Eager Test*, 31% for *Mystery Guest*, 16% for *Resource Optimism* and *Test Redundancy*. Then, in $\mathbf{RQ}_2$, we investigated how the anomaly detection approaches perform when compared to heuristic- and machine learning-based approaches, finding that anomaly detection approaches have a higher *Precision* and *Recall*, while the *F-Measure* is similar in the context of *Mystery Guest* and *Resource Optimism* detection. Our results and further analyses suggest that anomaly detection methods are promising solutions to be adopted in a real-world scenario. This has practical implications for developers who may benefit from using a combination of methods to optimize test smell detection. Moreover, we emphasized that the low precision of some methods may lead to an increased number of false positives, which developers should be aware of when interpreting the results. This is crucial for integrating such tools into real-world workflows without overwhelming developers with potentially irrelevant results.

[2] https://www.jetbrains.com/lp/devecosystem-2023/languages/

Furthermore, the low performance observed for the heuristic-based approaches suggests that their current definitions and implementations present challenges that limit their effectiveness.

The low *F-Measure* should be viewed not as a deterrent but as a starting point for further exploration and improvement of anomaly detection methods. Future research may explore the identification of new metrics to detect smelly instances and the feasibility of creating context-dependent test smell detectors, as well as the need to design more robust and accurate approaches. On the basis of the conclusions drawn in our study—especially those concerning the promises of context-dependent metrics and test smell detectors—we may provide some outlook on how research may approach the problem of test smell detection in the future. In particular, traditional metrics do not account for the specific context of the project or the development environment. Context-dependent metrics may consider various factors such as the project's domain, the team's coding practices, and the historical evolution of the test suite. By incorporating these factors, context-dependent metrics can provide more accurate and relevant assessments of test smells, tailoring the detection process to the unique characteristics of each project. For example, an adaptive algorithm could analyze the historical test execution data to identify patterns in how frequently different parts of the codebase are tested together. It could then dynamically adjust the criteria for detecting *Eager Test* instances, which occur when a test method unnecessarily tests multiple methods or functionalities at once. If a project has a pattern of integrating tightly coupled functionalities that are typically tested together, the algorithm would recognize this context and set a higher threshold for identifying *Eager Test* smells. Conversely, in a project where test methods are generally more focused and isolated, the algorithm would lower the threshold to catch potential issues more effectively.

This adaptive, context-aware approach may enhance the precision of test smell detection by aligning more closely with the project's actual coding practices and quality standards. Furthermore, anomaly detection methods can significantly benefit from incorporating context-dependent metrics. By understanding the specific norms and patterns within a project, these methods can better differentiate between normal and anomalous behaviors. This could involve integrating machine learning techniques that learn from the project's historical data and evolving practices, continuously refining the detection criteria based on the ongoing changes in the codebase and testing strategies. Additionally, investigating the impact of different contextual factors on the effectiveness of test smell detection can help refine these approaches, ensuring they are robust and generalizable across various types of projects and development environments.

Another area of investigation could be the integration of anomaly detection methods with traditional static analysis tools or more sophisticated semantic analysis approaches. Anomaly detection, which is typically used for identifying unusual patterns in data, can be employed to detect outliers in test code metrics. These outliers may possibly correspond to test smells or other quality issues. By combining anomaly detection with other tools that can better capture the context and intent behind test cases, the overall accuracy of the detection may improve. For example, combining code metric analysis with anomaly detection could help identify rare but critical test smells that are otherwise missed by conventional detectors.

In addition, extending validation efforts to diverse software systems (including the industry-scale projects) and languages would broaden the relevance and corroborate our findings in different contexts. Finally, we plan to empirically assess the anomaly detection approaches on other test smells to verify the scalability of our approach.

Acknowledgements

The authors would like to thank the handling editor and the anonymous reviewers for the insightful comments raised during the review process. These were instrument to improve the quality of our manuscript. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001; and FAPESB grants BOL0188/2020 and PIE0002/2022. Valeria is partially supported by the FWO SBO BaseCamp Zero project (Code: S000323N). This work has also been partially funded by the European Union under NextGenerationEU with the *RECHARGE* research project, which has been funded by MUR PRIN 2022 PNRR program (Code: P2022SELA7).

Declaration of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability Statement

The manuscript has data included as electronic supplementary material. In particular: datasets, detailed results, as well as scripts and additional resources useful for reproducing the study, are available as part of our online appendix on Figshare: `https://doi.org/10.6084/m9.figshare.24466336`.

Credits

Valeria Pontillo: Formal analysis, Investigation, Data Curation, Validation, Writing - Original Draft, Visualization. **Luana Martins**: Formal analysis, Investigation, Data Curation, Validation, Writing - Original Draft, Visualization. **Ivan Machado**: Conceptualization, Methodology, Validation, Writing - Review & Editing. **Fabio Palomba**: Conceptualization, Methodology, Validation, Writing - Review & Editing. **Filomena Ferrucci**: Supervision, Resources, Writing - Review & Editing.

References

[1] G. Catolino, F. Palomba, A. Zaidman, F. Ferrucci, Not all bugs are the same: Understanding, characterizing, and classifying bug types, Journal of Systems and Software 152 (2019) 165–181.

[2] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM computing surveys (CSUR) 41 (2009) 1–58.

[3] K. N. Neela, S. A. Ali, A. S. Ami, A. U. Gias, Modeling software defects as anomalies: A case study on promise repository., Journal of Software 12 (2017) 759–772.

[4] P. Afric, L. Sikic, A. S. Kurdija, M. Silic, Repd: Source code defect prediction as anomaly detection, Journal of Systems and Software 168 (2020) 110641.

[5] F. Lomio, L. Pascarella, F. Palomba, V. Lenarduzzi, Regularity or anomaly? on the use of anomaly detection for fine-grained jit defect prediction, in: 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), ACM, Pittsburgh, PA, USA, 2022, pp. 270–273.

[6] V. Kumar, Parallel and distributed computing for cybersecurity, IEEE Distributed Systems Online 6 (2005) 1–9.

[7] M. Ahmed, A. Naser Mahmood, J. Hu, A survey of network anomaly detection techniques, Journal of Network and Computer Applications 60 (2016) 19–31.

[8] A. Deursen, L. M. Moonen, A. Bergh, G. Kok, Refactoring Test Code, Technical Report, CWI (Centre for Mathematics and Computer Science), NLD, 2001.

[9] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, D. Binkley, An empirical analysis of the distribution of unit test smells and their impact on software maintenance, in: 2012 28th IEEE International Conference on Software Maintenance (ICSM), IEEE, 2012, pp. 56–65.

[10] G. Bavota, A. De Lucia, M. Di Penta, R. Oliveto, F. Palomba, An experimental investigation on the innate relationship between quality and refactoring, Journal of Systems and Software 107 (2015) 1–14.

[11] D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, A. Bacchelli, On the relation of test smells to software code quality, in: 2018 IEEE International Conference on Software Maintenance and Evolution, IEEE, 2018, pp. 1–12.

[12] G. Grano, F. Palomba, D. Di Nucci, A. De Lucia, H. C. Gall, Scented since the beginning: On the diffuseness of test smells in automatically generated test code, Journal of Systems and Software 156 (2019) 312–327.

[13] G. Grano, F. Palomba, H. C. Gall, Lightweight assessment of test-case effectiveness using source-code-quality indicators, IEEE Transactions on Software Engineering (2019).

[14] N. Koochakzadeh, V. Garousi, A tester-assisted methodology for test redundancy detection, Advances in Software Engineering 2010 (2010) 1–13.

[15] A. Peruma, K. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni, F. Palomba, Tsdetect: An open source test smells detection tool, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020, ACM, New York, NY, USA, 2020, p. 1650–1654.

[16] T. Virginio, L. Martins, L. Rocha, R. Santana, A. Cruz, H. Costa, I. Machado, Jnose: Java test smell detector, in: Proceedings of the 34th Brazilian Symposium on Software Engineering, ACM, New York, NY, USA, 2020, p. 564–569.

[17] W. Aljedaani, A. Peruma, A. Aljohani, M. Alotaibi, M. W. Mkaouer, A. Ouni, C. D. Newman, A. Ghallab, S. Ludi, Test smell detection tools: A systematic mapping study, in: Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering, EASE '21, ACM, New York, NY, USA, 2021, p. 170–180.

[18] D. Spadini, M. Schvarcbacher, A.-M. Oprescu, M. Bruntink, A. Bacchelli, Investigating severity thresholds for test smells, in: Proceedings of the 17th International Conference on Mining Software Repositories, MSR '20, ACM, New York, NY, USA, 2020, p. 311–321.

[19] L. Martins, C. Bezerra, H. Costa, I. Machado, Smart prediction for refactorings in the software test code, in: Brazilian Symposium on Software Engineering, ACM, New York, NY, USA, 2021, p. 115–120.

[20] M. Hadj-Kacem, N. Bouassida, A multi-label classification approach for detecting test smells over java projects, Journal of King Saud University-Computer and Information Sciences 34 (2021) 8692–8701.

[21] V. Pontillo, D. Amoroso d'Aragona, F. Pecorelli, D. Di Nucci, F. Ferrucci, F. Palomba, Machine learning-based test smell detection, Empirical Software Engineering 29 (2024) 1–44.

[22] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, D. Binkley, Are test smells really harmful? an empirical study, Empirical Software Engineering 20 (2015) 1052–1094.

[23] D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, J. C. Riquelme, Preliminary comparison of techniques for dealing with imbalance in software defect prediction, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, 2014, pp. 1–10.

[24] V. Pontillo, L. Martins, I. Machado, F. Palomba, F. Ferrucci, An empirical investigation into the capabilities of anomaly detection approaches for test smell detection — online appendix, 2024. URL: `https://doi.org/10.6084/m9.figshare.24466336`.

[25] E. Aleskerov, B. Freisleben, B. Rao, Cardwatch: A neural network based database mining system for credit card fraud detection, in: Proceedings of the IEEE/IAFE 1997 computational intelligence for financial engineering (CIFEr), IEEE, New York, NY, USA, 1997, pp. 220–226.

[26] C. Spence, L. Parra, P. Sajda, Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model, in: Proceedings IEEE workshop on mathematical methods in biomedical image analysis (MMBIA 2001), IEEE, Kauai, HI, USA, 2001, pp. 3–10.

[27] L. Tarassenko, P. Hayton, N. Cerneaz, M. Brady, Novelty detection for the identification of masses in mammograms, in: Fourth International Conference on Artificial Neural Networks, IET, Cambridge, UK, 1995, pp. 442–447.

[28] N. Sarafijanovic-Djukic, J. Davis, Fast distance-based anomaly detection in images using an inception-like autoencoder, in: P. Kralj Novak, T. Šmuc, S. Džeroski (Eds.), Discovery Science, Springer International Publishing, Cham, 2019, pp. 493–508.

[29] C.-L. Li, K. Sohn, J. Yoon, T. Pfister, Cutpaste: Self-supervised learning for anomaly detection and localization, in: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Nashville, TN, USA, 2021, pp. 9659–9669.

[30] F. Palomba, D. Di Nucci, A. Panichella, R. Oliveto, A. De Lucia, On the diffusion of test smells in automatically generated test code: An empirical study, in: Proceedings of the 9th International Workshop on Search-Based Software Testing, SBST '16, ACM, New York, NY, USA, 2016, p. 5–14.

[31] A. Peruma, K. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni, F. Palomba, On the distribution of test smells in open source android applications: An exploratory study, in: Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, CASCON '19, IBM Corp., USA, 2019, p. 193–202.

[32] L. Martins, H. Costa, I. Machado, On the diffusion of test smells and their relationship with test code quality of java projects, Journal of Software: Evolution and Process 1 (2023) e2532.

[33] N. Koochakzadeh, V. Garousi, Tecrevis: A tool for test coverage and test redundancy visualization, in: L. Bottaci, G. Fraser (Eds.), Testing – Practice and Research Techniques, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 129–136.

[34] S. Zhang, D. Jalali, J. Wuttke, K. Muslu, W. Lam, M. D. Ernst, D. Notkin, Empirically revisiting the test independence assumption, in: Proceedings of the 2014 International Symposium on Software Testing and Analysis, ISSTA 2014, ACM, New York, NY, USA, 2014, p. 385–396.

[35] J. Bell, G. Kaiser, E. Melski, M. Dattatreya, Efficient dependency detection for safe java test acceleration, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, ACM, New York, NY, USA, 2015, p. 770–781.

[36] M. Biagiola, A. Stocco, A. Mesbah, F. Ricca, P. Tonella, Web test dependency detection, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2019, ACM, New York, NY, USA, 2019, p. 154–164.

[37] G. Fraser, A. Gambi, J. M. Rojas, Teaching software testing with the code defenders testing game: Experiences and improvements, in: 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE, New York, NY, USA, 2020, pp. 461–464.

[38] A. Gyori, A. Shi, F. Hariri, D. Marinov, Reliable testing: Detecting state-polluting tests to prevent test dependency, in: Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, ACM, New York, NY, USA, 2015, p. 223–233.

[39] M. Greiler, A. Van Deursen, M.-A. Storey, Automated detection of test fixture strategies and smells, in: Software Testing, Verification and Validation (ICST), 2013, pp. 322–331.

[40] M. Greiler, A. Zaidman, A. v. Deursen, M.-A. Storey, Strategies for avoiding text fixture smells during software evolution, in: Working Conference on Mining Software Repositories, IEEE Press, San Francisco,

CA, USA, 2013, pp. 387–396.

[41] C. Huo, J. Clause, Improving oracle quality by detecting brittle assertions and unused inputs in tests, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, ACM, New York, NY, USA, 2014, p. 621–631.

[42] F. Palomba, A. Zaidman, A. De Lucia, Automatic test smell detection using information retrieval techniques, in: International Conference on Software Maintenance and Evolution, IEEE, Madrid, Spain, 2018, pp. 311–322.

[43] F. Pecorelli, G. Di Lillo, F. Palomba, A. De Lucia, Vitrum: a plug-in for the visualization of test-related metrics, in: Proceedings of the International Conference on Advanced Visual Interfaces, 2020, pp. 1–3.

[44] M. Taniguchi, S. Matsumoto, S. Kusumoto, Jtdog: A gradle plugin for dynamic test smell detection, in: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering, ASE '21, IEEE Press, New York, NY, USA, 2021, p. 1271–1275.

[45] Y. Yang, X. Hu, X. Xia, X. Yang, The lost world: Characterizing and detecting undiscovered test smells, ACM Transactions on Software Engineering and Methodology (2023).

[46] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, Springer Berlin, Heidelberg, 2012.

[47] V. Garousi, B. Küçük, Smells in software test code: A survey of knowledge in industry and academia, Journal of Systems and Software 138 (2018) 52–81.

[48] S. Lambiase, A. Cupito, F. Pecorelli, A. De Lucia, F. Palomba, Just-in-time test smell detection and refactoring: The darts project, in: Proceedings of the 28th International Conference on Program Comprehension, ACM, New York, NY, USA, 2020, p. 441–445.

[49] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto, A. De Lucia, Automatic test case generation: What if test code quality matters?, in: International Symposium on Software Testing and Analysis, ACM, 2016, pp. 130–141.

[50] S. Scalabrino, M. Linares-Vásquez, R. Oliveto, D. Poshyvanyk, A comprehensive model for code readability, Journal of Software: Evolution and Process 30 (2018) e1958.

[51] R. P. Buse, W. R. Weimer, Learning a metric for code readability, IEEE Transactions on software engineering 36 (2009) 546–558.

[52] F. Pecorelli, F. Palomba, A. De Lucia, The relation of test-related factors to software quality: a case study on apache systems, Empirical Software Engineering 26 (2021) 1–42.

[53] Y. Gil, G. Lalouche, On the correlation between size and metric validity, Empirical Software Engineering 22 (2017) 2585–2611.

[54] V. G. Yusifoğlu, Y. Amannejad, A. B. Can, Software test-code engineering: A systematic mapping, Information and Software Technology 58 (2015) 123–147.

[55] T. J. McCabe, A complexity measure, IEEE Transactions on software Engineering (1976) 308–320.

[56] G. Grano, C. De Iaco, F. Palomba, H. C. Gall, Pizza versus pinsa: On the perception and measurability of unit test code quality, in: 2020 IEEE international conference on software maintenance and evolution (ICSME), IEEE, 2020, pp. 336–347.

[57] G. Grano, S. Scalabrino, R. Oliveto, H. Gall, An empirical investigation on the readability of manual and generated test cases, in: Proceedings of the 26th International Conference on Program Comprehension, 2018, pp. 348–351.

[58] R. P. Buse, W. R. Weimer, Learning a metric for code readability, IEEE Transactions on Software Engineering 36 (2010) 546–558.

[59] J. Dorn, A general software readability model, MCS Thesis available from (http://www. cs. virginia. edu/weimer/students/dorn-mcs-paper. pdf) 5 (2012) 11–14.

[60] J. Hair Jr, J. F. Hair Jr, G. T. M. Hult, C. M. Ringle, M. Sarstedt, A primer on partial least squares structural equation modeling (PLS-SEM), Sage publications, 2021.

[61] S. Agrawal, J. Agrawal, Survey on anomaly detection using data mining techniques, Procedia Computer Science 60 (2015) 708–713.

[62] S. D. Palma, M. Mohammadi, D. Di Nucci, D. A. Tamburri, Singling the odd ones out: A novelty detection approach to find defects in infrastructure-as-code, in: Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation, MaLTeSQuE 2020, ACM, New York, NY, USA, 2020, p. 31–36.

[63] K.-L. Li, H.-K. Huang, S.-F. Tian, W. Xu, Improving one-class svm for anomaly detection, in: Proceedings of the 2003 International Conference on machine learning and cybernetics (IEEE Cat. No. 03EX693), volume 5, IEEE, Xi'an, 2003, pp. 3077–3081.

[64] F. T. Liu, K. M. Ting, Z.-H. Zhou, Isolation forest, in: Proceedings of the 8th IEEE International Conference on Data Mining, IEEE, Pisa, Italy, 2008, pp. 413–422.

[65] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Machine learning 63 (2006) 3–42.

[66] Z. Cheng, C. Zou, J. Dong, Outlier detection using isolation forest and local outlier factor, in: Proceedings of the Conference on Research in Adaptive and Convergent Systems, RACS '19, ACM, 2019, p. 161–168.

[67] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of Machine Learning Research 13 (2012) 281–305.

[68] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[69] R. Baeza-Yates, B. d. A. N. Ribeiro, et al., Modern information retrieval, ACM Press; Harlow, England: Addison-Wesley, New York, NY, USA, 2011.

[70] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, H. Nielsen, Assessing the accuracy of prediction algorithms for classification: an overview, Bioinformatics 16 (2000) 412–424.

[71] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in: kdd, 34, 1996, pp. 226–231.

[72] J. A. Hartigan, M. A. Wong, Algorithm as 136: A k-means clustering algorithm, Journal of the royal statistical society. series c (applied statistics) 28 (1979) 100–108.

[73] M. Breugelmans, B. Van Rompaey, Testq: Exploring structural and maintenance characteristics of unit test suites, in: WASDeTT-1: 1st International Workshop on Advanced Software Development Tools and Techniques, Citeseer, 2008, p. 11.

[74] M. R. Sheldon, M. J. Fillyaw, W. D. Thompson, The use and interpretation of the friedman test in the analysis of ordinal-scale data in repeated measures designs, Physiotherapy Research International 1 (1996) 221–228.

[75] P. B. Nemenyi, Distribution-free multiple comparisons, Princeton University, 1963.

[76] S. S. Shapiro, M. B. Wilk, An analysis of variance test for normality (complete samples), Biometrika 52 (1965) 591–611.

[77] K. R. Shahapure, C. Nicholas, Cluster quality analysis using silhouette score, in: 2020 IEEE 7th International Conference on data science and advanced analytics, IEEE, Sydney, NSW, Australia, 2020, pp. 747–748.

[78] A. Vysala, D. J. Gomes, Evaluating and validating cluster results, arXiv preprint arXiv:2007.08034 (2020) 37–47.

[79] A. Rosenberg, J. Hirschberg, V-measure: A conditional entropy-based external cluster evaluation measure, in: Proceedings of the 2007 joint Conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL), Prague, Czech Republic, 2007, pp. 410–420.

[80] S. Zhang, H.-S. Wong, Y. Shen, Generalized adjusted rand indices for cluster ensembles, Pattern Recognition 45 (2012) 2214–2226.

[81] F. Rahutomo, T. Kitasuka, M. Aritsugi, et al., Semantic cosine similarity, in: The 7th international student conference on advanced science and technology, volume 4, University of Seoul South Korea, 2012, p. 1.