Machine Learning-Based Test Smell Detection

Valeria Pontillo*, Dario Amoroso d'Aragona†, Fabiano Pecorelli†,
Dario Di Nucci*, Filomena Ferrucci*, Fabio Palomba*
vpontillo@unisa.it, dario.amorosodaragona@tuni.fi, fabiano.pecorelli@tuni.fi
ddinucci@unisa.it, fferrucci@unisa.it, fpalomba@unisa.it
*Software Engineering (SeSa) Lab — University of Salerno, Fisciano, Italy
†Tampere University — Tampere, Finland

***Abstract*—Context: Test smells are symptoms of sub-optimal design choices adopted when developing test cases. Previous studies have proved their harmfulness for test code maintainability and effectiveness. Therefore, researchers have been proposing automated, heuristic-based techniques to detect them. However, the performance of such detectors is still limited and dependent on thresholds to be tuned.**

Objective: We propose the design and experimentation of a novel test smell detection approach based on machine learning to detect four test smells.

Method: We plan to develop the largest dataset of manually-validated test smells. This dataset will be leveraged to train six machine learners and assess their capabilities in within- and cross-project scenarios. Finally, we plan to compare our approach with state-of-the-art heuristic-based techniques.

***Index Terms*—Test Smells; Test Code Quality; Machine Learning; Empirical Software Engineering.**

I. Introduction

Test cases are the first barrier against software faults, particularly during regression testing [38]. Development teams rely on their outcome to decide whether it is worth merging a pull request [24] or even deploying the system [7]. At the individual level, the developer's productivity is also partially dependent on the ability of tests to find real defects in production code [70] and the timely diagnosis of the underlying causes [48]. Unfortunately, when developing test cases, programmers may apply sub-optimal implementation choices that could introduce test debt [32], namely potential design problems that lead to unforeseen testing and debugging costs for developers [34]. Test smells, *i.e.*, symptoms of poor design or implementation choices in test code [63], represent one of the major sources of test debt [56], [62]. Several empirical studies have focused on test smells to understand their properties [62] and their impact on maintainability [5], [25], [59] and test effectiveness [26], by showing compelling evidence of the risks associated with the presence of test smells for software dependability.

For these reasons, researchers have investigated methods for automatically detecting test smells [22]. Such techniques discriminate tests affected (or not) by a certain type of smell by applying detection rules that compare the values of relevant metrics extracted from test code against some empirically identified thresholds. For instance, van Rompaey *et al.* [64] proposed a metric-based technique that computes several structural metrics (*e.g.*, number of production code calls made by a test case) and combines them into detection rules to highlight the likelihood of a test being smelly. A test is marked as smelly if the value overcomes a threshold.

Despite the effort spent by researchers so far, existing test smell detectors suffer from two key limitations. First and foremost, they have limited detection capabilities, behaving similarly to a random guessing approach [27], [42], [64]. Secondly, their performance is strongly influenced by the thresholds used in the detection rules to discriminate between smelly and non-smelly tests [20], [22]. These restrictions threaten the practical applicability of these approaches.

This registered report proposes the design of a novel test smell detector to overcome these limitations. The approach will be based on machine learning and employ structural and textual metrics to estimate the likelihood of a test being smelly. Besides avoiding the need to combine metrics using detection rules, a machine learning approach also avoids the problem of selecting thresholds, thus representing a promising solution to alleviate the limitations of heuristic-based techniques. Our approach is instantiated for the detection of four test smell types, *i.e.*, *Eager Test*, *Mystery Guest*, *Resource Optimism*, and *Test Redundancy*. Afterward, we propose an empirical evaluation plan to assess the performance of the devised detector on a new dataset of JAVA projects—which we will manually build, publicly releasing the largest manually-crafted dataset of test smells to date [22]—and compare its performance with three state-of-art heuristic-based techniques.

II. Related Work

Investigations on the design of test code were originally pointed out by Beck [6]. Van Deursen *et al.* [63] and Maszaros [37] defined catalogs of test smells along with their refactoring actions. More recently, Greiler *et al.* [27] devised TESTHOUND, a heuristic-based approach to identify six test smell types that was evaluated through semi-structured interviews. Palomba *et al.* [42] devised TASTE, a test smell detector which leverages textual metrics (*e.g.*, the conceptual cohesion of test methods [35]) to complement previous techniques and identify three test smell types. The detection rules proposed by Palomba *et al.* [42], were later implemented in DARTS [33], an INTELLIJ plugin that makes TASTE usable through a user interface. Peruma *et al.* [49] proposed TSDETECT, a test smell detector that identifies 19 test smell types, including *Assertion Roulette*, *Eager Test*, and *Lazy Test*. Pecorelli *et al.* [44]

proposed VITRuM, a Java plugin to provide developers with static and dynamic test-related metrics and identify seven test smell types. Similarly, Wang *et al.* [65] proposed PyNose, a Python plugin to detect 17 test smells. Koochakzadeh *et al.* [30] proposed TeReDetect, a tool that uses rules and dynamic metrics to detect *Test Redundancy*, *i.e.*, a test that could be removed without impacting the test suite. De Bleser *et al.* [17] proposed SoCRATES, a fully automated tool that combines syntactic and semantic data to identify six test smells in Scala software systems. Our paper is complementary to these researches since it introduces a new, orthogonal method based on machine learning to identify test smells that will not require tuning thresholds. Furthermore, we plan to conduct a large-scale empirical study on a manually-validated dataset, making our investigation the largest ever done in test smell detection research. Other related work concerns the empirical analyses of test smells. Tufano *et al.* [62] investigated the lifecycle of test smells, while Bavota *et al.* [5] showed that test smells are highly diffused in software projects and impact the understandability of test code. Similar results were later achieved when considering automatically generated test cases [25] and in software systems developed using the combination of Scala and ScalaTest [16]. Furthermore, Spadini *et al.* [59] showed that test smells impact the maintainability of both test and production code. Spadini *et al.* [58] also discovered that test-driven code reviews might help developers discover design flaws in test code. All these studies serve as motivation for our paper.

Based on the empirical evidence provided in the past, test smells represent a relevant threat to software reliability that should be promptly detected. We aim to employ machine learning (ML) algorithms, which have been previously used for code smell detection—the interested reader may find a comprehensive literature analysis on code smells in [2]. Although code and test smells share a similar high-level definition, they do not share the same characteristics. It is, therefore, worth analyzing the main differences we expect compared to the previous researches on code smell detection. According to the literature available, ML-based code smell detection comes with three significant limitations concerning (i) data imbalance, (ii) subjectivity of code smell data, and (iii) a set of predictors that poorly contribute to the accuracy of the detection [46].

As for the data imbalance limitation, previous literature has shown that test smells are more diffused than code smells, *e.g.*, Bavota *et al.* [5] found *Eager Test* instances to affect around 35% of test classes. Conversely, code smells typically affect a meager percentage of classes (*i.e.*, around 2%) [41]. Therefore, we think the limitation of data imbalance could have a lower significance when dealing with test smells. Nevertheless, we planned to investigate the use of data balancing to understand whether this additional step could benefit the models.

Concerning subjectivity, we envision a strong relationship between test and code smells. The dataset we will build may suffer from the subjectivity of the authors who will make the validation; therefore, we will also employ external developers, as explained in Section IV-C.

As for the predictors, we will rely on metrics adopted by existing heuristic techniques. While we applied the same strategy as previously done for code smell detection [46], heuristic techniques for test smell detection are typically more accurate than those dealing with code smell detection [1]. Therefore, we expect the performance of our models not to be strongly influenced by this limitation.

III. Goals and Research Questions

The *goal* of the study is to evaluate the extent to which machine learning is suitable for test smell detection, with the *purpose* of improving test code quality by removing detrimental design flaws. The *perspective* is of researchers and practitioners interested in understanding the performance and limitations of machine learning techniques for test smell detection. Specifically, our paper is structured around four research questions (**RQ**s).

RQ$_1$. *Which are the features that provide more information gain to a machine learning-based test smell detector?*

RQ$_2$. *What is the performance of a machine learning-based test smell detector?*

RQ$_3$. *How does a machine learning-based test smell detector perform compared to heuristic-based approaches?*

With the first research question (**RQ$_1$**), we seek to understand which metrics contribute the most to the detection of test smells. These observations will be used to (i) quantify the predictive power of metrics and (ii) identify the most promising features to include in our machine learning approach. In **RQ$_2$** we run our machine learning approach against a manually-validated oracle of test smells (described later in this section) to quantify its detection performance capabilities. Afterward, with **RQ$_3$** we aim to compare the performance of our technique with the one achieved by state-of-the-art approaches based on heuristics: such validation will allow us to understand the actual value of a machine learning approach, *i.e.*, should it work worse than heuristic approaches, its usefulness would be limited, as practitioners might still found heuristic approaches more beneficial. Last but not least, we plan for an additional *in-vivo* evaluation of the capabilities of the machine learning model. Should the results coming from **RQ$_2$** and **RQ$_3$** be sufficiently promising, we aim to investigate the extent to which the predictions made are useful for developers to diagnose and/or refactor test smells. Hence, the last research question will be:

RQ$_4$. *What is the practitioners' perception of the test smells output by a machine learning-based test smell detector?*

To design and report our empirical study, we will follow the empirical software engineering guidelines by Wohlin et al. [68], other than the *ACM/SIGSOFT Empirical Standards*.[1]

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

IV. Dataset Construction

The first step of our investigation will be the creation of a manually-validated dataset of test smells.

A. Projects Selection

We will first collect test data from a dataset of 70 open-source Java projects, publicly available on GitHub, and 51,549 test cases. These projects are part of a larger, popular dataset known as the International Dataset of Flaky Tests (IDoFT).[2] The selection is driven by two main factors. First, we consider the entire set of test cases contained in these projects, *i.e.*, not only those labeled as flaky, to complement IDoFT with additional information related to test smells. In this way, researchers will be provided with a unique database containing various test code-related issues, which would be beneficial to stimulate further research on test code quality. These projects are highly diverse: they have different characteristics, scopes, and sizes. For the lack of space, more detailed statistics on those projects are available in our online appendix [52]. Secondly, the rationale for using this dataset comes from previous observations made by Pontillo *et al.* [53]. In their study, the authors ran a state-of-the-art test smell detector named VITRuM [44] and identified a high number of test smells, *i.e.*, they found that around 80% of test cases are smelly. While we will not use automated tools to collect test smell data, the high diffuseness of test smells in the dataset suggests that it may be worth manually analyzing them.

B. Selecting Test Smells

We have already performed a comprehensive literature analysis to extract all the test smells automatically detectable by the current techniques. We started from the list of test smell detection tools reported in a systematic mapping study by Aljedaani *et al.* [1]. This study reports all the test smell detection tools available in the literature and the test smells they detect. From an initial set of 22 tools, we included only those (i) supporting Java as a programming language, as the vast majority of tools use only Java as the target language, and (ii) relying on a metric-based approach, since machine learning classifiers require a set of metrics to be used as predictors. This filtering phase led us to a final number of ten tools.

Afterward, we analyzed each tool and extracted information about the test smells they detect and the metrics they use for the detection. As a machine learning-based classification would be meaningless if based on a single metric, we decided to include only test smells for which at least two metrics have been defined (more details about the metrics are reported in Section V-B). This led us to the selection of a set of six test smell types, namely *Empty Test*, *Eager Test*, *Mystery Guest*, *Sensitive Equality*, *Resource Optimism*, and *Test Redundancy*.

However, we noticed that detecting two of these smells was very trivial (*i.e.*, *Empty Test* and *Sensitive Equality*); therefore, the use of a machine learning-based approach would not lead to any detection performance improvement. *Empty Test* is defined as *"A test method that is empty or does not have executable statements"*; thus, a heuristic approach could objectively identify test cases that suffer from this issue. The same consideration also applies to *Sensitive Equality*, which occurs when *"an assertion has an equality check by using the toString method"*. Based on the above consideration, we decided to discard these two test smells, resulting in a final set of four test smells reported in Table I together with their definition. Another discussion point concerns the *Resource Optimism* smell. Given its definition, it is likely that information-flow or dynamic analyses might be potentially more suitable for detecting it. In this sense, a machine learning solution might be sub-optimal, yet we aim at experimenting with the extent to which it may provide valuable insights to detect the smell. These observations might be used to understand how the performance of machine learning compares to existing approaches and, perhaps, be later used by researchers to combine it with novel, more precise information flows or dynamic sources of information.

C. Collecting Test Smell Data

Considering the unavailability of a reliable dataset containing information about test smell occurrences, we plan to manually identify them in the dataset described in Section IV-A. The first two authors of the paper (from now on "the inspectors") will conduct the entire process to mitigate the subjectivity of the validation. Given the impracticability of manually analyzing all 51,549 test cases, the process will be conducted on a statistically significant stratified sample of 12,550 test cases (confidence level = 99%, margin of error = 1%). The sample will be stratified based on the total number of test cases in the considered projects. In this way, we will analyze a sample that keeps the same proportion of test cases of the original population, *i.e.*, a larger project will account for more tests than a smaller one.

As a first step, both inspectors will independently analyze a subset of 1,255 test methods (equal to 10% of the total)—a third inspector (*i.e.*, the third author of the paper) will be in charge of making the final decision about the disagreements. Then, the results of this first validation will be compared through Cohen's κ statistic [15], which measures the *inter-rater agreement* of the inspection task. Through this measure, the inspectors will understand whether their manual analyses converge toward a common procedure, leading to an objective assessment. This step will be repeated, using different subsets, until a strong agreement is achieved [36]. As a final step of the validation process, the unclassified instances will be equally split between the two inspectors. This process will allow us to develop and deliver the largest manually-validated collection of test smells over 12,550 test cases. We will release the dataset as a publicly available source so that other researchers might exploit it to build on top of our findings.

While the formal process described above is supposed to mitigate possible bias when labeling the smelliness of test code, this may still contain subjective test smell instances. For this reason, we will also plan to involve experienced developers to validate the test smells affecting the considered

[2] https://mir.cs.illinois.edu/flakytests/

tests. Since it seems unreasonable to ask for external validation of the entire set of 12,550 test cases (it would be excessively costly in terms of time and effort required by external developers), we will proceed as follows. We will randomly select a subset of 628 test cases (5% of the test cases that will be validated) and involve 200 external developers through the PROLIFIC platform,[3] a research instrument to select research participants. Through the appropriate filters, we will involve experienced developers. The developers will be provided with a definition of the test smells subject of the study and will be asked to assess the smelliness of four test cases. Note that, having 628 tests and 200 developers, we will be able to perform cross-checking, *i.e.*, several developers will assess a subset of 172 test cases to verify the consistency among the evaluations provided. In doing so, we will also collect background information about the participants. Should the external validation results be in line with or close enough to the internal one, we will consider the dataset construction completed. Otherwise, we will extend the external validation to the entire set of test cases by involving up to 600 developers recruited through PROLIFIC.

V. Machine Learning-based Test Smell Detection

This section illustrates the envisioned machine learning-based approach for test smell detection.

A. Dependent Variable

Our goal consists of automatically detecting the presence of test smells in test code components. Therefore, as a dependent variable, we will rely on a binary value indicating the presence/absence of a specific test smell type. We will consider as a dependent variable the outcome of the validation process discussed in Section IV-C.

B. Independent Variables

To collect a set of reliable predictors for each test smell under consideration, we will use the metrics from heuristic approaches already available in the literature. Specifically, while performing the process described in Section IV-B for the inclusion of test smells, we collected all the metrics defined and used by the available detection approaches. Table I reports the list of metrics used for classifying each test smell with their description. Our online appendix [52] also includes references to all the tools relying on the same metrics. However, we are aware that these metrics might not represent a complete set of features characterizing test smells, *i.e.*, there might be additional metrics not considered by previous work that could contribute to identifying the four test smells. We will exploit the experience gained while manually identifying test smells to search for additional patterns and metrics that may be used as independent variables. We will reserve the right to define new metrics that complement the existing ones.

C. Selecting Machine Learning Algorithms

Our work proposes the first machine learning-based test smell detector; therefore, the most suitable classifier is still unknown. We will experiment with a set of classifiers belonging to different families that have been widely used in problems related to software maintenance and evolution [10]–[12], [18], [45], [46]. The goal is to (i) understand which machine learning algorithm is the best for test smell detection and (ii) increase the generalizability of the results. In details, we will evaluate *Decision Tree* [21], *Naive Bayes* [19], *Multilayer Perceptron* [61], and *Support Vector Machine* [40], as basic classifier. We will also consider two ensemble techniques, such as *Ada Boost* [57] and *Random Forest* [9].

D. Configuration and Training

When training the selected machine learners, we will experiment with multiple under- and over-sampling techniques to balance our data. We will compare them as further reported in Section VI-B. As for the under-sampling, we will consider the use of NEARMISS 1, NEARMISS 2, and NEARMISS 3 algorithms [69]. Finally, we will experiment with a RANDOM UNDERSAMPLING approach that randomly explores the distribution of majority instances and under-samples them. As for the over-sampling, we will experiment with *Synthetic Minority Over-sampling Technique*, a.k.a SMOTE [13], and advanced versions of this algorithm, *i.e.*, *Adaptive Synthetic Sampling Approach*, a.k.a ADASYN [29] and the BORDERLINE-SMOTE [28]. We will also experiment with a RANDOM OVERSAMPLING approach that randomly explores the distribution of the minority class and over-samples them.

Finally, concerning the classifiers configuration, we will experiment with the hyper-parameters of the classifiers using the RANDOM SEARCH strategy [8]: this search-based algorithm randomly samples the hyper-parameters space to find the best combination of hyper-parameters maximizing a scoring metric (*i.e.*, the Matthews Correlation Coefficient). We plan to develop the entire pipeline with the SCIKIT-LEARN library [47] in PYTHON.

E. Validation of the Approach

To assess the performance of our models, we will perform within- and cross-project validation. These validations aim to quantify the performance of the models in two different scenarios. We are interested to understand (i) how accurate can the performance be when a test smell detection model is trained using data of the same project where it should be applied and (ii) how accurate is the model when trained using external data to the project where it should be applied.

For the within-project validation, we will perform a stratified 10-fold cross-validation [60]—we will apply it to individual projects. This strategy randomly partitions the data into ten folds of equal size, allowing us to maintain the correct proportion in every split between smelly and non-smelly instances. It iteratively selects a single fold as a test set, while the other nine are used as a training set.

[3] PROLIFIC website: https://www.prolific.co/.

TABLE I
TEST SMELLS INCLUDED IN OUR STUDY, THEIR DEFINITION AND THE INDEPENDENT VARIABLES FOR EACH SMELL UNDER INVESTIGATION.

Test Smell	Definition	Metric	Description
Eager Test	A test method that invokes many methods of the object being tested.	NMC	Number of Method Calls
		PTMI	Number of Production Types Method Invocations
		PET	Probability of a Method to be affected by Eager Test based on its textual content
Mystery Guest	A test that uses external resources (*e.g.*, databases or files).	NRF	Number of References to Files
		NRDB	Number of References to Database
Resource Optimism	A test that uses external resources without checking the state of these.	ERNC	External Resource state (not files) Not Checked
		FRNC	File Resource state Not Checked
Test Redundancy	A test that could be removed without impacting the test suite.	PR	Pair Redundancy is the ratio between the items covered by a test and those covered by another one
		SR	Suite Redundancy is the ratio between the items covered by a test compared and those covered by all others tests in the test suite

For the cross-project validation, we will adopt the *Leave-One-Out Cross-Validation* strategy [55], a special case of K-fold cross-validation with K equal to N, the number of projects in the set. We will train models using the test cases of $N-1$ projects and use the test cases of the remaining project as the test set. The process will be repeated N times to ensure that each project will occur in the test set once.

VI. EXECUTION PLAN

A. ***RQ***$_1$ *- In Search of Suitable Metrics for Machine Learning-Based Test Smell Detection*

Finding a set of metrics to characterize the four considered test smells represents a first challenge to face [39]. As explained in Section IV-B, we will start focusing on the metrics that have been used by previous researchers when detecting test smells. In other words, we will investigate whether a machine learning solution is suitable to combine structural and textual metrics that were considered in isolation by previous work. Table I lists and describes each considered test smell. These metrics capture the smelliness of tests under different perspectives, taking into account the size of fixtures and test suites, cohesion and coupling aspects of tests, and conceptual relationships between the methods composing test suites.

We will quantify the predictive power of each metric by computing their *information gain* [54]. This step will be used as a *probing* method, *i.e.*, it will estimate the contribution provided by each metric other than acting as a feature selection instrument: we will indeed use as predictors the metrics having an information gain higher than zero, i.e., we will discard the metrics that do not provide any expected beneficial effect on the performance. More specifically, the output of the information gain algorithm consists of a ranked list where the features of the model are placed in a descending manner, meaning that those contributing the most are placed at the top. We will employ the *Gain Ratio Feature Evaluation* algorithm [54] available in the SCIKIT-LEARN library [31].

B. ***RQ***$_2$ *- Assessing the Performance of Our Machine Learning-Based Test Smell Detector*

When assessing the performance of our models, we will proceed with a stepwise analysis of the various components included in the experimentation. We will perform an *ablation* study to analyze the contribution of each configuration and training step to the overall models' performance. We will experiment with multiple combinations, *e.g.*, we will experiment how the performance varies when including (and not) the feature selection step, the data balancing, and the hyper-parameter optimization, other than considering the performance variations given by the adoption of different validation procedures. In this way, we will also be able to assess the best possible pipeline for the problem of test smell detection.

To evaluate the performance of the various combinations experimented and address **RQ**$_2$, we will compute a number of state-of-the-art metrics such as *precision*, *recall*, *F-Measure* [3], *Matthews Correlation Coefficient* (MCC) [4], and the *Area Under the Curve - Precision-Recall (AUC-PR).*

To support the results achieved, we will statistically verify the validity of the findings. We will use the Wilcoxon test [67], with 0.05 as a significance value, computed on the distributions of MCC values of machine learning-based and heuristic-based techniques over the different projects and the different test smell types. We will also rely on Cliff's Delta (or d), a non-parametric effect size measure [14] to assess the magnitude of the measured differences.

C. ***RQ***$_3$ *- Comparing Machine Learning- and Heuristic-Based Techniques for Test Smell Detection*

To complement the analysis of the performance of our machine learning-based approach, we plan to conduct a benchmark study to compare our approach with state-of-the-art techniques based on heuristics. On the one hand, we will assess the real usefulness of our approach: should our model be less performing than the baselines, its practical use would be limited. On the other hand, we will measure the extent to which our technique overcomes existing approaches, thus understanding the strengths and weaknesses of our approach compared to existing detectors. We will compare our approach against three heuristic-based baselines:

TSDETECT [49]. We select this tool as it represents the current state of the art in test smell detection [1] and, at the same time, it is able to detect the highest number of test smell types. Out of the four test smells included in our study, TSDETECT can identify three of them, *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()`.

TEREDETECT [30]. We select this tool as it is the only one available to detect *Test Redundancy* smell instances. The tool detects the smell by computing code coverage and analyzing whether two tests cover similar paths.

DARTS [33]. The model built for *Eager Test* relies on an information retrieval metric (*i.e.*, PET). For this reason, we believe it might be worth to compare the model against an information retrieval-based heuristic technique, which is the one implemented within DARTS [33]. The tool relies on the detection rule proposed by Palomba *et al.* [42]. As such, it detects *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.

To enable a fair comparison, we will run the heuristic approaches against the same systems considered in **RQ**$_2$. None of these heuristic tools require configuration, *i.e.*, they can be run against the source code without the need of specifying any parameter: this ensures the execution of their original implementations, hence avoiding possible bias due to wrong configuration of the tools. We will employ the same evaluation metrics used to assess the machine learning models, *i.e.*, *precision*, *recall*, *F-Measure*, *MCC*, and *AUC-PR*. Similarly to **RQ**$_2$, we will also statistically verify the validity of the findings between our and baseline techniques by using the Wilcoxon Test [67] and the Cliff's Delta [14].

D. ***RQ***$_4$ *- In-Vivo Evaluation of the Machine Learning-Based Test Smell Detector*

After assessing the performance of the machine learning-based approach through lab experimentation and comparison with the state of the art, we will then consider the definition of a new empirical analysis aiming at verifying the capabilities of the model in the wild. More specifically, we plan to experiment the machine learning models against the test code of open-source systems not included in the dataset. We will (i) collect the 50 most popular GitHub projects - according to their number of stars; (ii) apply the models against the test code of those projects to detect test smells; (iii) open new issues on their issue tracker, one for each test smell detected; (iv) assess how developers react to these pieces of information. Should these predictions be used by developers to discuss about the quality of their tests or to refactor them, this would imply that our model is effective in practice. We will compute metrics such as the number of comments per issue, the number of refactoring actions performed by developers, and the number of closed issues. In addition, we will also report on the developer's opinions expressed in the comments. With this *in-vivo* evaluation we aim at (i) measuring the extent to which the predictions of the models are actually usable and useful for developers to identify and remove potential issues in test code; and (ii) reducing possible threats to generalizability. In any case, it is worth remarking that such an evaluation would be worth to be conducted *if and only if* the performance of the models would be high enough; otherwise, the in-vivo validation would become an unnecessary waste of the time for developers, who would be dealing with noise in their issue trackers. Hence, we will decide on whether to pursue the experiment based on the F-Measure achieved in **RQ**$_2$: should this be higher than 70%, we will proceed with the evaluation.

E. *Publication of Generated Data*

All the data generated from our study will be publicly available in an online repository [52]. We also plan to release the scripts, other than the data collected and used for the statistical analysis that we will present in the paper.

VII. LIMITATIONS

A first possible limitation will concern the dataset exploited in our study. Starting from a publicly available dataset containing 51,549 test cases, we will manually identify test smells on a stratified sample of 12,550 tests. Such a manual inspection will represent the main threat to the validity of our conclusions. To mitigate this threat, we plan to involve more inspectors who will follow a systematic approach. First, they will independently analyze subsets of test cases and repeatedly compute inter-rater agreement measures to verify to what extent the manual inspection will converge toward a common procedure. Only after this step, they will analyze the remaining unclassified test cases. Whenever needed, a third inspector will also help solve disagreements. In addition, we also plan to involve external, experienced developers within an additional manual validation of a sample of test cases. Should it not align with our internal validation, we will proceed with a full external validation based on the developers' assessment.

In **RQ**$_3$, we selected alternative state-of-the-art heuristic approaches which have been employed by the research community, showing significant results (*e.g.*, like in the case of TSDETECT [50], [51] and DARTS [33]), or the only available tool for the detection of *Test Redundancy* (*i.e.*, TEREDETECT [30]). In this respect, it is worth remarking that none of them require configuration; therefore, we can rely on their original implementations. Other limitations may involve the implementation of our machine learning-based detector. From a methodological standpoint, we will conduct *probing* and *ablation* studies to identify the most relevant independent variables and the most appropriate pipeline to devise the models. As such, possible threats to the creation of the models will be mitigated by analyzing multiple aspects that might influence the results (*e.g.*, which features to consider and how to train the classifier). We believe that the procedures we will follow are precise enough to ensure the validity of the study. From a technical perspective, we will rely on the SCIKIT-LEARN library [47], thus leveraging all its structures and algorithms. While we cannot exclude possible implementation errors, the development community around SCIKIT-LEARN usually tests source code appropriately.

Concerning the relationship between treatment and outcome, we will exploit a set of widely-used metrics to evaluate the experimented techniques (*i.e.*, precision, recall, F-measure, MCC, AUC-PR), and we will provide qualitative examples to show the differences between the compared approaches. When assessing the contribution of the features to use in our approach, we will rely on the *Gain Ratio Feature Evaluation* algorithm [54], which the research community has widely used for the same purpose [10], [12], [43]. Finally, we will use appropriate statistical tests, *i.e.*, the Wilcoxon Test and the Cliff's Delta, which will allow us to support our findings.

In terms of generalizability, we will validate the devised models in both within- and cross-project scenarios to assess the capabilities of the models at large. Especially with the cross-project validation, we expect to provide insights into the generalizability of the results. At the same time, should the performance of the devised models be sufficiently high, we will experiment with them with the test code of open-source systems not included in the dataset. Such an *in-vivo* evaluation is in line with the *lab-to-field generalization* strategy proposed by Wieringa and Daneva [66]. In this generalization strategy, a technology (*e.g.*, the prediction models) is first experimented with in-house (*e.g.*, through within- and cross-project validation) and then assessed in the field where it is supposed to be used (*e.g.*, any arbitrary open-source systems). Nonetheless, we still recognize the presence of additional threats to generalizability. These are mainly connected to (i) our focus on JAVA projects, which is due to the availability of tools working for this programming language [1] and (ii) the large, but still limited, number of projects that we will consider. In this respect, we can claim for the *generalizing by similarity* principle described by Ghaisas *et al.* [23]: it is likely that similar results might be obtained in projects having similar characteristics to those analyzed in our work.

VIII. CONCLUSION

The ultimate goal of our research is to define a machine learning-based test smell detection and compare its performance with those of heuristic baselines. We will start working toward this goal by creating the largest manually-validated dataset of test smells: we will report information about the presence of four smell types over 12,550 test cases. As part of our future work, we plan to assess our technique following the methodology described in Section VI.

ACKNOWLEDGEMENT

Fabio gratefully acknowledges the support of the Swiss National Science Foundation through the SNF Projects No. PZ00P2_186090. This work has been partially supported by the EMELIOT national research project, funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY).

REFERENCES

[1] W. Aljedaani, A. Peruma, A. Aljohani, M. Alotaibi, M. W. Mkaouer, A. Ouni, C. D. Newman, A. Ghallab, and S. Ludi. Test smell detection tools: A systematic mapping study. *Evaluation and Assessment in Software Engineering*, pages 170–180, 2021.

[2] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang. Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Information and Software Technology*, 2019.

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

[4] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424, 2000.

[5] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley. Are test smells really harmful? an empirical study. *Empirical Software Engineering*, 20(4):1052–1094, 2015.

[6] K. Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[7] M. Beller, G. Gousios, and A. Zaidman. Oops, my tests broke the build: An explorative analysis of travis ci with github. In *Int.l Conf. on Mining Software Repositories (MSR)*, pages 356–367. IEEE, 2017.

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

[9] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[10] G. Catolino, D. Di Nucci, and F. Ferrucci. Cross-project just-in-time bug prediction for mobile apps: An empirical assessment. In *Int.l Conf. on Mobile Software Engineering and Systems*, pages 99–110. IEEE, 2019.

[11] G. Catolino and F. Ferrucci. An extensive evaluation of ensemble techniques for software change prediction. *Journal of Software: Evolution and Process*, page e2156, 2019.

[12] G. Catolino, F. Palomba, A. De Lucia, F. Ferrucci, and A. Zaidman. Enhancing change prediction models using developer-related factors. *Journal of Systems and software*, 143:14–28, 2018.

[13] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[14] N. Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, 114(3):494, 1993.

[15] J. Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[16] J. De Bleser, D. Di Nucci, and C. De Roover. Assessing diffusion and perception of test smells in scala projects. In *Int.l Conf. on Mining Software Repositories*, pages 457–467. IEEE Press, 2019.

[17] J. De Bleser, D. Di Nucci, and C. De Roover. Socrates: Scala radar for test smells. In *ACM SIGPLAN Symposium on Scala*, pages 22–26. ACM, 2019.

[18] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, and A. De Lucia. A developer centered bug prediction model. *IEEE Transactions on Software Engineering*, 2017.

[19] R. O. Duda, P. E. Hart, et al. *Pattern classification and scene analysis*. A Wiley-Interscience publication. Wiley, 1973.

[20] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo. A review-based comparative study of bad smell detection tools. In *Int.l Conf. on Evaluation and Assessment in Software Engineering*, page 18. ACM, 2016.

[21] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *icml*, volume 99, pages 124–133. Citeseer, 1999.

[22] V. Garousi and B. Küçük. Smells in software test code: A survey of knowledge in industry and academia. *Journal of systems and software*, 138:52–81, 2018.

[23] S. Ghaisas, P. Rose, M. Daneva, K. Sikkel, and R. J. Wieringa. Generalizing by similarity: Lessons learnt from industrial case studies. In *Int.l Workshop on Conducting Empirical Studies in Industry (CESI)*, pages 37–42. IEEE, 2013.

[24] G. Gousios, A. Zaidman, M. Storey, and A. Van Deursen. Work practices and challenges in pull-based development: the integrator's perspective. In *Int.l Conf. on Software Engineering-Volume 1*, pages 358–368. IEEE Press, 2015.

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

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

[27] M. Greiler, A. Van Deursen, and M.-A. Storey. Automated detection of test fixture strategies and smells. In *Software Testing, Verification and Validation (ICST)*, pages 322–331, 2013.

[28] H. Han, W. Wang, and B. Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *Int.l Conf. on intelligent computing*, pages 878–887. Springer, 2005.

[29] H. He, Y. Bai, E. A. Garcia, and S. Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *Int.l joint Conf. on neural networks (IEEE world congress on computational intelligence)*, pages 1322–1328. IEEE, 2008.

[30] N. Koochakzadeh and V. Garousi. A tester-assisted methodology for test redundancy detection. *Advances in Software Engineering*, 2010, 2010.

[31] O. Kramer. Scikit-learn. In *Machine learning for evolution strategies*, pages 45–53. Springer, 2016.

[32] P. Kruchten, R. L. Nord, and I. Ozkaya. Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21, 2012.

[33] S. Lambiase, A. Cupito, F. Pecorelli, A. De Lucia, and F. Palomba. Just-in-time test smell detection and refactoring: The darts project. In *Int.l Conf. on Program Comprehension*, pages 441–445, 2020.

[34] E. d. S. Maldonado and E. Shihab. Detecting and quantifying different types of self-admitted technical debt. In *Int.l Workshop on Managing Technical Debt (MTD)*, pages 9–15. IEEE, 2015.

[35] A. Marcus and D. Poshyvanyk. The conceptual cohesion of classes. In *Int.l Conf. on Software Maintenance*, pages 133–142. IEEE, 2005.

[36] M. L. McHugh. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276–282, 2012.

[37] G. Meszaros. *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.

[38] G. J. Myers, C. Sandler, and T. Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[39] K. K. Nicodemus and J. D. Malley. Predictor correlation impacts machine learning algorithms: implications for genomic studies. *Bioinformatics*, 25(15):1884–1890, 2009.

[40] W. S. Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[41] F. Palomba, D. Di Nucci, A. Panichella, R. Oliveto, and A. De Lucia. On the diffusion of test smells in automatically generated test code: An empirical study. In *Int.l Workshop on Search-Based Software Testing*, pages 5–14. ACM, 2016.

[42] F. Palomba, A. Zaidman, and A. De Lucia. Automatic test smell detection using information retrieval techniques. In *Int.l Conf. on Software Maintenance and Evolution*, pages 311–322. IEEE, 2018.

[43] F. Palomba, M. Zanoni, F. A. Fontana, A. De Lucia, and R. Oliveto. Toward a smell-aware bug prediction model. *IEEE Transactions on Software Engineering*, 2017.

[44] F. Pecorelli, G. Di Lillo, F. Palomba, and A. De Lucia. Vitrum: A plug-in for the visualization of test-related metrics. In *AVI 2020*, pages 1–3, 2020.

[45] F. Pecorelli, D. Di Nucci, C. De Roover, and A. De Lucia. On the role of data balancing for machine learning-based code smell detection. In *ACM SIGSOFT Int.l workshop on machine learning techniques for software quality evaluation*, pages 19–24, 2019.

[46] F. Pecorelli, F. Palomba, D. Di Nucci, and A. De Lucia. Comparing heuristic and machine learning approaches for metric-based code smell detection. In *Int.l Conf. on Program Comprehension*, pages 93–104. IEEE Press, 2019.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] A. Perez, R. Abreu, and A. van Deursen. A test-suite diagnosability metric for spectrum-based fault localization approaches. In *Int.l Conf. on Software Engineering*, pages 654–664. IEEE Press, 2017.

[49] A. Peruma, K. Almalki, C. D. Newman, M. W. M., A. Ouni, and F. Palomba. Tsdetect: An open source test smells detection tool. In *ACM Joint Meeting on European Software Engineering Conf. and Symposium on the Foundations of Software Engineering*, pages 1650–1654, 2020.

[50] A. Peruma, K. S. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni, and F. Palomba. On the distribution of test smells in open source android applications: An exploratory study. 2019.

[51] A. Peruma, C. D. Newman, M. W. Mkaouer, A. Ouni, and F. Palomba. An exploratory study on the refactoring of unit test files in android applications. In *Int.l Conf. on Software Engineering Workshops*, pages 350–357, 2020.

[52] V. Pontillo, D. Amoroso D'Aragona, F. Pecorelli, D. Di Nucci, F. Ferrucci, and F. Palomba. Machine learning-based test smell detection — online appendix. https://github.com/darioamorosodaragona-tuni/ML-Test-Smell-Detection-Online-Appendix.

[53] V. Pontillo, F. Palomba, and F. Ferrucci. Toward static test flakiness prediction: a feasibility study. In *Int.l Workshop on Machine Learning Techniques for Software Quality Evolution*, pages 19–24, 2021.

[54] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[55] P. Refaeilzadeh, L. Tang, and H. Liu. Cross-validation. In *Encyclopedia of database systems*, pages 532–538. Springer, 2009.

[56] G. Samarthyam, M. Muralidharan, and R. K. Anna. Understanding test debt. In *Trends in Software Testing*, pages 1–17. Springer, 2017.

[57] R. E. Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[58] D. Spadini, F. Palomba, T. Baum, S. Hanenberg, M. Bruntink, and A. Bacchelli. Test-driven code review: an empirical study. In *Int.l Conf. on Software Engineering*, pages 1061–1072. IEEE Press, 2019.

[59] D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, and A. Bacchelli. On the relation of test smells to software code quality. In *2018 IEEE Int.l Conf. on Software Maintenance and Evolution*, pages 1–12. IEEE, 2018.

[60] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society: Series B (Methodological)*, 36(2):111–133, 1974.

[61] H. Taud and J. Mas. Multilayer perceptron (mlp). In *Geomatic Approaches for Modeling Land Change Scenarios*, pages 451–455. Springer, 2018.

[62] M. Tufano, F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk. An empirical investigation into the nature of test smells. In *Int.l Conf. on Automated Software Engineering*, pages 4–15, 2016.

[63] A. Van Deursen, L. Moonen, A. van den Bergh, and G. Kok. Refactoring test code. In *Int.l Conf. on extreme programming and flexible processes in software engineering (XP2001)*, pages 92–95, 2001.

[64] B. Van Rompaey, B. Du Bois, S. Demeyer, and M. Rieger. On the detection of test smells: A metrics-based approach for general fixture and eager test. *IEEE Transactions on Software Engineering*, 33(12):800–817, 2007.

[65] T. Wang, Y. Golubev, O. Smirnov, J. Li, T. Bryksin, and I. Ahmed. Pynose: A test smell detector for python. In *Int.l Conf. on Automated Software Engineering (ASE)*, pages 593–605. IEEE, 2021.

[66] R. Wieringa and M. Daneva. Six strategies for generalizing software engineering theories. *Science of computer programming*, 101:136–152, 2015.

[67] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

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

[69] S. Yen and Y. Lee. Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset. In *Intelligent Control and Automation*, pages 731–740. Springer, 2006.

[70] Y. Zhang and A. Mesbah. Assertions are strongly correlated with test suite effectiveness. In *Joint Meeting on Foundations of Software Engineering*, pages 214–224. ACM, 2015.