Smart Prediction for Test Smell Refactorings

Luana Martins
Institute of Computing
Federal University of Bahia
Salvador, Brazil
martins.luana@ufba.br

Heitor Costa
Dept. of Computer Science
Federal University of Lavras
Lavras, Brazil
heitor@ufla.br

Fabio Palomba
Dept. of Computer Science
University of Salerno
Fisciano, Italy
fpalomba@unisa.it

Ivan Machado
Institute of Computing
Federal University of Bahia
Salvador, Brazil
ivan.machado@ufba.br

ABSTRACT

Software plays a critical role in modern society, making software testing essential for ensuring quality. Poor testing practices, known as test smells, reduce test code quality and maintainability. Although refactoring is a widely used technique to address these issues, little is known about when developers choose to refactor test smells and whether these actions actually improve test code quality. This study introduces a machine learning approach to guide test smell refactoring. First, we aim to mine refactorings performed by developers to derive a catalog of test-specific refactorings. Findings show testing framework evolution helps address test smells. Second, we aim to understand whether developers target low-quality test codes to perform refactorings and their effects on code quality. Our findings reveal that developers tend to refactor structurally low-quality test code more often. Third, we aim to learn whether developers would perform refactorings and which refactorings they would apply to improve the test code quality. Results show that Support Vector Machine models predicted refactoring decisions with 30–100% accuracy. ***This paper is a summary of the full paper version [8] of the doctoral thesis [7].***

KEYWORDS

Test Refactoring, Test Smells, Test Code Quality, Machine Learning

1 Introduction

Software testing is a fundamental activity for software quality assurance and consists of developing helpful test cases to find bugs caused by code changes [6]. However, given the complexity of the software under test, the lack of expertise, and time pressure can lead software developers to use bad practices to either design or implement the test code, also known as test smells [4, 13, 15]. The presence of test smells can negatively affect the test code quality, harming the software testing and maintenance activities [2, 3, 11, 14].

To fix test smells, developers should refactor the test code in a way that does not change the test logic [5]. While most refactoring recommendation tools target the production code, little attention has been devoted to detecting test smells and refactoring the test code [1]. Prior test smells detection and refactoring strategies, based on rules and metrics, were simplistic and not derived from common practices used by developers [12]. In particular, those strategies mostly rely on predefined thresholds or rules to detect test smells. However, deciding whether a refactoring operation fixes a particular test smell requires developers' expertise and intuition [14]. Furthermore, existing tools [1] only assist in detection and require manual intervention for refactoring, and they do not offer multiple refactoring alternatives for a given test smell.

Therefore, it is important for developers to have proper guidelines for identifying test smells and refactoring the test code. This aligns with this study proposal that aims to analyze supervised Machine Learning (ML) algorithms to classify developers' intentions in applying test refactoring and the specific test refactoring operation. This research contributes to Computer Science by integrating Software Engineering and Machine Learning to improve test code maintenance through automated refactoring support. It introduces a *catalog of test-specific refactorings* to support developers' decision making on whether and how to refactor test code. Additionally, it investigates the *relationship between test code quality and refactoring* likelihood, demonstrating that low-quality test code is more prone to refactoring. Finally, it applies *ML models to predict developers' refactoring decisions* to improve the overall quality of the test code.

In summary, the contributions of this study might positively impact the developers' effectiveness on testing and maintenance activities (*social aspect*), the development of automated tools to assist in test refactoring (*technical aspect*), the understanding of refactoring practices and their effects on software quality (*scientific aspect*), and the efficiency of software development, potentially reducing time and resource consumption (*environmental aspect*).

2 Research Statement and Research Questions

In this work, we investigated ML techniques to classify the developers' intention to apply test refactoring and which test-specific refactoring operations they would apply. In particular, we investigated three main Research Questions (RQ):

> **RQ$_1$.** *How do developers perform test code refactorings to fix test smells in open-source projects?*

In **RQ$_1$**, our objective was to catalog test code refactorings commonly performed by developers in practice to address test smells. To achieve this, we conducted a qualitative study [9] on a sample of open-source projects to manually classify test code changes. Additionally, we investigated developers' preferences regarding refactoring operations for fixing test smells by submitting pull requests to open-source projects.

> **RQ$_2$.** *How do test refactoring operations affect test code quality?*

In **RQ$_2$**, our objective was twofold: i) to determine whether low-quality test classes, as measured by structural metrics and the presence of test smells, are more likely to have test refactorings; ii) to assess the impact of test code refactorings on test code quality. We conducted an empirical study [10] involving the analysis of refactorings carried out by developers.

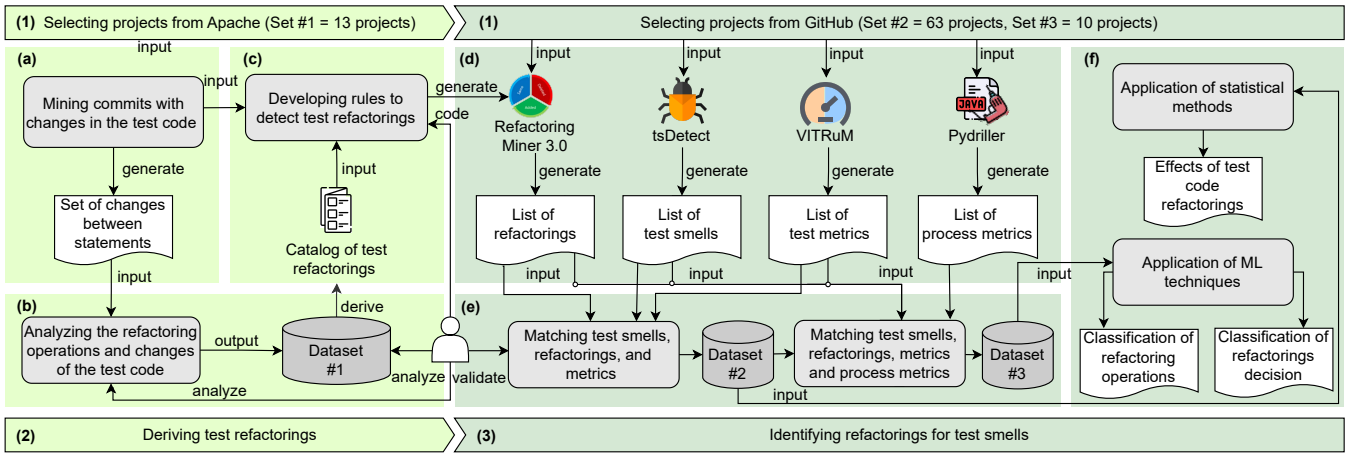

Figure 1: Overview of our approach.

RQ$_3$. *How accurately can we suggest test refactoring operations for fixing test smells using ML techniques?*

In **RQ$_3$**, our objective was to explore the performance of ML algorithms in classifying developers' intentions regarding test refactoring. This investigation encompassed two classification tasks: i) classifying code changes where developers would apply refactorings; ii) classifying the specific test refactorings.

3 Approach Design

Figure 1 illustrates our three-step approach, described as follows.
(1) Selecting subject projects: This step consisted of selecting software projects from GitHub. The *Set #1* comprises 13 projects from the Apache Foundation, and we used it to perform a manual analysis of test code. The *Set #2* comprises 63 projects, and we used it for repository mining to gather data on test smells, refactorings, and structural metrics using automated tools. The *Set #3* comprises ten projects with information on test smells, refactorings, structural and process metrics. We used it to train and evaluate ML models.
(2) Deriving test refactorings: This step consisted of extracting code changes from *Set #1* (step a) and manually classifying them into test smells and test refactorings (step b), resulting in *Dataset #1*. The outcome of this process is a catalog to perform test refactoring (step c). This catalog proved instrumental in developing rules to enhance existing tools for detecting test refactorings (refer to Section 4).
(3) Identifying refactoring operations for test smells: This step consisted in collecting data using tools (step d), generating *Dataset #2* and *Dataset #3* (step e). Finally, we analyzed the data from both datasets (step f). We used *Dataset #2* to analyze the impacts of test code refactorings on test quality (refer to Section 5), and *Dataset #3* to identify refactoring opportunities and specific operations developers would perform in the code (refer to Section 6).

4 Mining Test Refactorings in Practice (RQ1)

This section presents a summary of our empirical study [9] to answer **RQ$_1$**. The *goal* of our empirical study was to analyze the test refactoring operations performed by developers in practice, with the *purpose* of creating a catalog of the test smells developers deem as problematic and which test refactoring operations developers apply to fix the test smells.

We manually found nine test smells being refactored in the 13 open-source projects. In particular, two test-specific refactorings are associated with the upgrade of JUnit version targeting two test smells (*Handling Exception* and *Assertion Roulette*). Other nine version-agnostic refactorings were identified for seven test smells (*Inappropriate Assertion*, *Assertion Roulette*, *Bad Naming*, *Ignored Test*, *Empty Test*, *Unknown Test*, and *Sleepy Test*). While many of these test-specific refactorings are documented in the literature, this analysis unveils new refactorings aimed at dealing with the *Inappropriate Assertion*, e.g., developers misuse the *assertEquals* to verify whether a method call returns true instead of using *assertTrue*.

To validate the proposed catalog, we submitted 13 pull requests to refactor 143 test smells. As a result, developers' feedback suggests a general appreciation for the efforts to improve test quality. In addition, the developers often provided justifications not only for why certain test smells existed but also for why certain refactorings were not worth the change, which offered historical or functional insights that might take time to be apparent from data analysis.

5 Test Refactorings Impact on Quality (RQ2)

This section addresses this knowledge gap by summarizing our exploratory empirical study [10] to answer **RQ$_2$**. The *goal* of the empirical study was to analyze the test refactoring operations performed by developers over the history of software projects, with the *purpose* of understanding (1) whether low-quality test classes provide indications on which test classes are more likely of being refactored, and (2) as a consequence, to what extent test refactorings are effective in improving quality of test classes.

Our first analysis aimed to assess whether refactoring operations are more likely to be observed on test classes exhibiting test code quality concerns. The results show that the *Assertion Density* metric is related to most test refactorings. It indicates that the more assertions in the test code, the more likely developers would refactor it. In addition, we found some unexpected results when analyzing whether the presence of test smells drives the test refactorings.

Our second analysis aimed at investigating the effects of test refactorings on quality metrics and test smells. The results show that most values remained the same before and after applying test refactorings. However, we have some particular cases. The *Extract Class* refactoring improved the test classes concerning the *Lines of Code*, *Number of Methods*, and *Weight Method Class*. Differently, after applying the *Extract Method* refactoring, the *Number of Methods* and *Weight Method Class* metrics increased their values. As for the test-specific refactorings, the *Assertion Density* metric presents a negative effect after applying the *Parameterized Test*, *Replace Test annot. w/ assertThrows* and *Replace Rule annot. w/ assertThrows*.

6 Developer-Oriented Test Refactoring Recommendations (RQ3)

This section presents our investigation of the performance of supervised ML algorithms in classifying the developers' intention to apply test refactoring, answering **RQ**$_3$. Our *goal* was to investigate the effectiveness of supervised ML algorithms in classifying the code changes where practitioners apply test refactoring actions, considering two levels of granularity: i) the classification of code changes where practitioners apply a refactoring in test classes and ii) the classification of the specific test refactoring action applied by practitioners in a certain code change. Both levels have the *purpose* of studying the feasibility of an automated instrument for test quality assurance, which practitioners can use to identify test classes requiring maintenance effort and assess how to improve them.

First, we explored how accurately supervised ML algorithms classify the code changes where developers would apply test refactoring. We evaluated the performance of six ML algorithms (*Decision Trees*, *Naive Bayes, Logistic Regression, Extra-Tree, Support Vector Machine,* and *Random Forest*) to classify test code changes in which practitioners would do test refactoring using test smells, test metrics, and process variables, i.e., code churn, as predictors. We ran and analyzed 540 models to classify the code changes in test classes. As a result, we found out that the Support Vector Machine algorithm outperforms the others; the performance varies from 30% to 100%.

After analyzing the performance of ML models in classifying the developer's intention to perform test refactoring, we verified the performance of classifying a specific test code refactoring operation. Before proceeding, it is important to highlight a consideration that reduced the number of refactorings analyzed. First, we ran 540 models for each test refactoring, for a total of 27,000 models. Once we had collected the results, we observed that for 32 refactorings, not enough information was found to train our models due to the low number of true instances. On the remaining 18 refactorings, for 11 we were able to gather information from only one project, for 4 cases from two projects, and for the last three refactorings, i.e., *Modify Method Annotation*, *Remove Method Annotation*, and *Add Method Annotation*, we gathered information from four, six, and seven projects, respectively. When we classify specific test refactoring operations, performance decreases due to a lower amount of data. The F-measure obtained varies between 19% and 60%.

7 Conclusions

This study investigated how developers refactor the test code to fix test smells in practice and what factors drive developers to refactor the test code. In addition, we explored ML techniques to classify whether developers would apply refactorings in the test code and which test-specific refactoring operations they would apply. By considering the developers' perspective, this research lays the foundation for more tailored refactoring techniques that can enhance the effectiveness of test-specific refactoring recommendation tools.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, Fundação de Amparo a Pesquisa do Estado da Bahia (FAPESB) grants BOL0188/2020 and PIE0002/2022, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) grant 312195/2021-4, Fundação de Amparo a Pesquisa do Estado de Alagoas (FAPEAL) grants 60030.0000000462/2020 and 60030.0000000161/2022. Fabio is supported by the Swiss National Science Foundation through the SNF Project No. PZ00P2_186090 (TED).

REFERENCES

[1] Wajdi Aljedaani, Anthony Peruma, Ahmed Aljohani, Mazen Alotaibi, Mohamed Wiem Mkaouer, Ali Ouni, Christian D. Newman, Abdullatif Ghallab, and Stephanie Ludi. 2021. Test Smell Detection Tools: A Systematic Mapping Study. In *Evaluation and Assessment in Software Engineering* (Trondheim, Norway). ACM, New York, NY, USA, 170–180.

[2] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea Lucia, and Dave Binkley. 2015. Are Test Smells Really Harmful? An Empirical Study. *Empirical Software Engineering* 20, 4 (Aug. 2015), 1052–1094.

[3] Denivan Campos, Larissa Rocha, and Ivan Machado. 2021. Developers perception on the severity of test smells: an empirical study. In *Iberoamerican Conference on Software Engineering* (Virtual Event). arxiv, Costa Rica, 1–14. arXiv:2107.13902

[4] Arie Deursen, Leon M. Moonen, A. Bergh, and Gerard Kok. 2001. *Refactoring Test Code.* Technical Report. Centre for Mathematics and Computer Science, NLD.

[5] Martin Fowler. 1999. *Refactoring - Improving the Design of Existing Code.* Addison-Wesley, Upper Saddle River, NJ.

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

[7] Luana Martins. 2024. *Smart prediction for test smell refactorings.* Ph. D. Dissertation. Universidade Federal da Bahia. https://repositorio.ufba.br/handle/ri/39886

[8] Luana Martins, Heitor Costa, Fabio Palomba, and Ivan Machado. 2025. Smart prediction for test smell refactorings. In *Anais estendidos do XVI Congresso Brasileiro de Software: Teoria e Prática - Concurso de Teses e Dissertações em Engenharia de Software* (Recife/PE). SBC, Porto Alegre, RS, Brasil, 15.

[9] Luana Martins, Taher A Ghaleb, Heitor Costa, and Ivan Machado. 2024. A comprehensive catalog of refactoring strategies to handle test smells in Java-based systems. *Software Quality Journal* 32, 2 (2024), 641–679.

[10] Luana Martins, Valeria Pontillo, Heitor Costa, Filomena Ferrucci, Fabio Palomba, and Ivan Machado. 2025. Test code refactoring unveiled: where and how does it affect test code quality and effectiveness? *Empirical Software Engineering* 30, 1 (2025), 1–39.

[11] Fabio Palomba, Dario Di Nucci, Annibale Panichella, Rocco Oliveto, and Andrea De Lucia. 2016. 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* (Austin, Texas). ACM, New York, NY, USA, 5–14.

[12] Anthony Peruma, Steven Simmons, Eman Abdullah AlOmar, Christian D Newman, Mohamed Wiem Mkaouer, and Ali Ouni. 2022. How do i refactor this? An empirical study on refactoring trends and topics in Stack Overflow. *Empirical Software Engineering* 27, 1 (2022), 1–43.

[13] Nildo Silva Junior, Larissa Rocha Soares, Luana Martins, and Ivan Machado. 2020. A survey on test practitioners' awareness of test smells. *CoRR* abs/2003.05613 (2020), 1–14. arXiv:2003.05613 https://arxiv.org/abs/2003.05613

[14] Davide Spadini, Martin Schvarcbacher, Ana-Maria Oprescu, Magiel Bruntink, and Alberto Bacchelli. 2020. Investigating Severity Thresholds for Test Smells. In *Proceedings of the 17th International Conference on Mining Software Repositories* (Seoul, Republic of Korea). ACM, NY, USA, 311–321.

[15] Michele Tufano, Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. 2016. An Empirical Investigation into the Nature of Test Smells. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore). ACM, New York, NY, USA, 4–15.