Students vs. Professionals: Improving the Learning of Software Testing

Zhongyan Chen
zhongyan.chen@manchester.ac.uk
The University of Manchester
Manchester, UK

ABSTRACT

Software testing is a crucial phase of software development. Educators now are assessing tests written by students. Methods have been proposed to assess the completeness of student-written tests. However, there is more to good test quality than completeness, and these additional quality are not assessed by the previous work. Test smells are patterns of poorly designed tests that may negatively affect the quality of test and production code. We propose to assess test smells in students' code to improve the quality of software tests they write. In the early stages of this research, we will examine whether practitioners actually spend time and energy to remove those test smells summarized by researchers backed by real-world software projects. Then, we will develop and evaluate our new assessment method.

KEYWORDS

test smell, software unit test, computer science education

ACM Reference Format:
Zhongyan Chen. 2022. Students vs. Professionals: Improving the Learning of Software Testing. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3510454.3517058

1 INTRODUCTION

Software testing has a vital role in software system development as it reveals hidden defects in software systems and assures their qualities. Software testing uses approximately 40%-50% of total resources, 30% of total effort and 50%-60% of the total cost of software developments [11]. To cope with the increasing importance of software testing, computer science educators have been assigning software testing coursework along with regular programming exercises and assessing the quality of both production and test code written by students. An approach for assessing students' test code is using automated tools as they have advantages like speed, availability, and consistency [2].

Current automated assessment tools use three common methods to examine the quality of student-written tests: code coverage, mutation analysis, and all-pairs comparison. Functionally, all three methods focus on assessing the completeness of the students' test suites. However, test quality has more aspects than completeness. Other qualities are not assessed by the above methods. Thus, new methods are needed for assessing these additional qualities, thereby improving the learning outcome.

Test smells are poorly designed software test codes that may undermine overall software test quality and even inject bugs, causing higher development costs in the future. Since good design and programming practices a range of software test quality, teaching such concepts explicitly has the potential to help computer science students be aware of those previously less emphasized aspects and thus write better quality tests.

The goal of my PhD is to improve the learning outcomes of software engineering students in universities by developing tools to support the addition of test smells to teaching and assessment. In the first year, I'll focus on test smells in real-world software systems, aiming to renew the knowledge of test smells in an industrial context. The second year of my PhD will focus on student analytics: identifying student-specific test smells and proposing methods to measure test code quality based on findings so far. Tools for assessment and feedback will also be proposed. Finally, the evaluation of proposed methods will be conducted in the third year.

2 RELATED WORK

Several methods have been proposed for evaluating test suites written by students [1, 6, 10]. Code coverage examines the percentage of production code that is covered by the tests. It can provide feedback effectively, but it has limitations regarding evaluating the adequacy of a test suite. For instance, a high coverage test suite cannot ensure the production code's correctness nor the completeness of implementation. Mutation analysis is a test assessment method where production code is injected with small incorrect changes, known as mutants. The quality of the test suite is assessed by calculating the percentage of detected mutants [5]. This approach is more capable of assessing the adequacy of tests than code coverage, but it costs too much time and computational resources for educational purposes. Finally, the all-pairs method runs every student's test against each other's production code. Although Edwards and Shams argued that this method is the best among all three methods [7], it has limited applicability. For example, it cannot be used when students are asked to implement systems with different functionalities. None of the three methods support assessment of non-functional properties of software tests beyond completeness.

Test smells in education have received less attention than other directions so far. Bai et al. compared students' and professionals' perceptions about unit testing[3]. Buffardi and Aguirre-Ayal analyzed the relationship between three test smells and test accuracy

ICSE '22 Companion, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9223-5/22/05…$15.00
https://doi.org/10.1145/3510454.3517058

in students' work [4]. We might expect that test smells may be introduced more often by students due to lack of experience. Currently, to the best of our knowledge, no research has been published on exploring student-specific test smells nor proposing tools for teaching or assessing test smells in classrooms.

There has been substantial research focusing on the study of test smells. During the last two decades, researchers have identified a set of test smells [9]. In terms of approaches to deal with test smells, various automated tools have been presented to detect test smells [12, 13, 15, 19]. Other research was focused on the consequences of test smells as well as refactoring methods to remove them [9]. An exploratory study concluded that some test smells are positively correlated with the existence of faults in the production code [14]. Van Bladel and Demeyer referred to removing test smells as a part of test code refactoring [17], and Van Deursen et al. summarized a set of bad test smells and ways to remove them [18]. They provided theoretical and technical foundations for assessing students' work by detecting test smells in their code and giving test smell related feedback.

Although previous research has provided strong foundations for this topic, there are still gaps between researchers and practitioners. Questions like whether the test smells documented in research papers exist in actual code bases and whether practitioners spend effort removing those test smells need to be answered. Interests should be shifted to the test smells existing in real-world code bases to avoid the waste of resources if the well-studied test smells are proven to be absent or insignificant in the test suites of actual software projects. Answering the above questions should filter some test smells that are outdated or unimportant in the current industrial context, thereby providing students with state-of-the-art knowledge.

3 RESEARCH HYPOTHESIS

My PhD project is formed by two parts: the preliminary study of test smells made by professionals and the proposal of a new teaching and assessment method. The preliminary study is planned to be completed during the first year of my PhD. The reason for conducting it was explained in section 2. With the findings of the preliminary study, we will confirm that the knowledge to be taught aligns with modern industry. Then, we propose and evaluate our teaching and assessment method.

The hypothesis for the preliminary study is proposed as follow:

- **Hypothesis 1**: Some well-studied test smells no longer widely exist in recent software projects. Even if some do, practitioners wish not to spend time and effort removing them.

The hypothesis for later stages of my PhD research is:

- **Hypothesis 2**: Some test smells are made specifically by students.
- **Hypothesis 3**: Assessing test smells and providing relevant feedback to students will help them avoid or remove test smells. Thus improves the quality of software tests written by students.

Since verifying hypothesis 2 and 3 partly depends on the findings of the preliminary study, it could be premature to form research questions for them. To validate hypothesis 1, a set of research questions (RQ) are raised:

RQ1:Which test smells supported by state-of-the-art proposed tools exist in real code bases?

The pool of "well-studied test smells" and their definitions are provided by Peruma et al. [13], as it proposed one of the latest JUnit test smell detection tools during the time of writing.

RQ2: What standard test smells are seen as sufficiently to be routinely removed/introduced by developers?

The goal of this RQ is to gain insights of test smells that developers care about, allowing us to filter insignificant test smells.

RQ3: Do developers wish to remove test smells in their systems, and why?

This RQ is designed for gathering developers' motivation for their test refactoring. It is expected that their opinions are needed to verify hypothesis 1.

4 EXPECTED CONTRIBUTION

The goal of my PhD is to help computer science students to write better quality tests in terms of recognizing and preventing test smells. New assessment and feedback methods will also be proposed to support the goal. The expected contributions are:

1) to re-evaluate the meaningfulness of well-studied test smells in real-world projects.

2) to discover some student-specific test smells.

3) to propose a new method of assessing the qualities of software tests written by students.

5 PLAN OF EVALUATION

The first year of my PhD aims to answer all RQs from section 3. Initially, a small-scale pilot experiment will be commenced to examine the tsDetect [13] tool and to understand what information may be needed from practitioners. After implementing the feedback obtained from the pilot experiment, including understanding the ethical application and GDPR requirements, the formal experiment will be conducted. With different aims, we plan to adapt silva's methodology in [16]. It includes selecting GitHub repositories, monitoring refactoring commits, interviewing author practitioners and analyzing practitioners' responses. Practitioners' opinions will be gathered via interviews, a survey, or through analysis of git commit comments.

The later years of my PhD will be focused on proposing new methods of teaching and assessment of software testing based on the findings of my first year. With the revised set of test smells, new student analytical tools will be developed to detect test smells in code written by students available from a previous study [8]. Meanwhile, the concept of test smells will be introduced in upcoming runs of software engineering classes in the Department of Computer Science, the University of Manchester. By mining the code, student-specific test smells are expected to be revealed. The impact of introducing the concept of test smells to student-written tests will be evaluated by comparing the prevalence of the test smells in code from new students' test code with the previous students.

REFERENCES

[1] Kalle Aaltonen, Petri Ihantola, and Otto Seppälä. 2010. Mutation analysis vs. code coverage in automated assessment of students' testing skills. In *Proceedings*

of the ACM international conference companion on Object oriented programming systems languages and applications companion. 153–160.

[2] Kirsti M Ala-Mutka. 2005. A survey of automated assessment approaches for programming assignments. *Computer science education* 15, 2 (2005), 83–102.

[3] Gina R Bai, Justin Smith, and Kathryn T Stolee. 2021. How Students Unit Test: Perceptions, Practices, and Pitfalls. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 248–254.

[4] Kevin Buffardi and Juan Aguirre-Ayala. 2021. Unit Test Smells and Accuracy of Software Engineering Student Test Suites. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 234–240.

[5] Richard A DeMillo, Richard J Lipton, and Frederick G Sayward. 1978. Hints on test data selection: Help for the practicing programmer. *Computer* 11, 4 (1978), 34–41.

[6] Stephen H Edwards. 2004. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*. 26–30.

[7] Stephen H Edwards and Zalia Shams. 2014. Comparing test quality measures for assessing student-written tests. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 354–363.

[8] Sukru Eraslan, Kamilla Kopec-Harding, Caroline Jay, Suzanne M Embury, Robert Haines, Julio César Cortés Ríos, and Peter Crowther. 2020. Integrating GitLab metrics into coursework consultation sessions in a software engineering course. *Journal of Systems and Software* 167 (2020), 110613.

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

[10] Michael H Goldwasser. 2002. A gimmick to integrate software testing throughout the curriculum. *ACM SIGCSE Bulletin* 34, 1 (2002), 271–275.

[11] Divya Kumar and Krishn Kumar Mishra. 2016. The impacts of test automation on software's cost, quality and time to market. *Procedia Computer Science* 79 (2016), 8–15.

[12] Stefano Lambiase, Andrea Cupito, Fabiano Pecorelli, Andrea De Lucia, and Fabio Palomba. 2020. Just-In-Time Test Smell Detection and Refactoring: The DARTS Project. In *Proceedings of the 28th International Conference on Program Comprehension*. 441–445.

[13] Anthony Peruma, Khalid Almalki, Christian D Newman, Mohamed Wiem Mkaouer, Ali Ouni, and Fabio Palomba. 2020. 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*. 1650–1654.

[14] Abdallah Qusef, Mahmoud O Elish, and David Binkley. 2019. An exploratory study of the relationship between software test smells and fault-proneness. *IEEE Access* 7 (2019), 139526–139536.

[15] Railana Santana, Luana Martins, Larissa Rocha, Tássio Virgínio, Adriana Cruz, Heitor Costa, and Ivan Machado. 2020. RAIDE: a tool for Assertion Roulette and Duplicate Assert identification and refactoring. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*. 374–379.

[16] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. 2016. Why we refactor? confessions of github contributors. In *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering*. 858–870.

[17] Brent van Bladel and Serge Demeyer. 2017. Test refactoring: a research agenda. In *Proceedings SATToSE*.

[18] Arie Van Deursen, Leon Moonen, Alex Van Den Bergh, and Gerard Kok. 2001. Refactoring test code. In *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*. Citeseer, 92–95.

[19] Tongjie Wang, Yaroslav Golubev, Oleg Smirnov, Jiawei Li, Timofey Bryksin, and Iftekhar Ahmed. 2021. PyNose: A Test Smell Detector For Python. *arXiv preprint arXiv:2108.04639* (2021).