Investigating Developer Perception on Test Smells Using Better Code Hub
- Work in Progress -

Martin Schvarcbacher[1,2], Davide Spadini[2,3], Magiel Bruntink[2] and Ana Oprescu[1]

[1]University of Amsterdam
[2]Software Improvement Group
[3]Delft University of Technology

Abstract

Test smells can be found in test code using a variety of tools. In this paper, we present our integration of a test smell detection tool into Better Code Hub (BCH), an online environment for monitoring code quality and identifying problems in it. We extended BCH with test smell detection and observe how developers react to various instances of test smells. By integrating this detection into BCH, we gain access to a wide range of developers working on both open-source and commercial projects using their own code. We study whether developers consider these test smells important and what they do with them in future code changes. From our preliminary results, we found out a high test smell detection accuracy for most test smells; however, developers are only willing to remove a small portion of them.

1 Introduction

The majority of the developer focus is spent on writing and improving production code quality, while test code quality is often not prioritized [14]. This can lead to the creation of so called test smells [3], which can make the tests hard to modify and can decrease their effectiveness. Test smells can be hard to detect by manual inspection [14], which means that we have to rely on automatic test smell detection to find such instances. Removing test smells can have a positive impact on the test suite quality, reduce the test flakiness and discover bugs not previously covered by these faulty unit tests [9].

SIG, a consultancy company with the headquarter in Amsterdam (NL), developed a tool to analyze GitHub repositories code quality called Better Code Hub (BCH)[1]. BCH checks the GitHub codebase against 10 easy to follow software engineering guidelines. Currently the guideline for test quality in BCH only include assertions density and test code LOC. We extended the existing test quality metrics with test smell detection and integrated it into BCH to do our research on test smells.

Existing research on test smells focuses on detection of test smells [8, 12, 13, 2, 5] with detection accuracy and recall often surpassing 95%. The impact of test smells on code maintainability and test effectiveness was studied in [2, 11].

We aim to answer the following research questions:

RQ1: What is the perception of developers on test smells in their codebase?

RQ2: Which test smells developers consider to be important?

Our research on test smells is different in the following aspects: (1) we investigate all instances of test smells on code the developers have previously interacted with; (2) by using Better Code Hub, we can

In: Anne Etien (eds.): Proceedings of the 12th Seminar on Advanced Techniques Tools for Software Evolution, Bolzano, Italy, July 8-10 2019, published at http://ceur-ws.org

[1] `https://bettercodehub.com/`

cover more projects and users to gain a more diverse data set; (3) by giving developers concrete examples from their own codebase, we expect the developers to be aware of the context in which both the production and test code was developed and how the test smell was created. Furthermore, for each detected test smell in the code base, we ask the developers (through a survey) their opinion on the importance of the detected smell. Because the survey is integrated directly in BCH, we are able to invite users outside of SIG to take part in the survey and also analyze their own code.

After conducting our experiment, we found that developers recognize test smell instances once they are presented to them with high accuracy. However, depending on the test smell, they are less willing to refactor the test suite to remove this test smell even for test smells which are rated as having an high impact on software maintainability.

The paper is structured as follows: section 2 contains the related works on test smell research, section 3 presents the research questions and experiment design, section 4 provides the results followed by section 5 with an analysis of the results, followed by our future plans in section 6 and concludes with section 7.

2 Background and Related Work

The concept of test smells originates from the work of van Deursen et al. [3] and test smells have been expanded with several other instances [10, 7]. Garousi et al. [4] performed a study on the existing literature and categorized all of the published test smells and their detection methods. We aim to study test smells in both open-source and closed-source projects, as there might be differences in how these two projects are developed and maintained [16]. Most of the existing research on test smells involves using open-source projects and the evaluation is often performed by people not involved with the project, such as [11, 6].

Currently there is ongoing research into the discovery and classification of test code smells and their impact on software quality and reliability [1, 15, 9]. Being able to detect test code smells and point those out to the developers can help them refactor the test suite to be less flaky and catch more defects. Currently there are many tools to detect test smells [8, 12, 13, 2, 5]; however, none of them are integrated into a modern tool used by developers.

3 Experiment Design

3.1 Tools used

We first modified Better Code Hub[2] (BCH) using a Chrome extension to display different metrics for test code quality than the current production build. The "Automate Tests" metric results were modified to include the detected test smells in the analyzed source code along with the test smell type. The modification is shown in Figure 1. The user can then click on the individual smelly method and see the source code. There is a possibility to submit their feedback on whether the test code segment is, according to their opinion, a valid instance of the test smell. This feedback can be given for each found test smell. A sample view of the code view and feedback form is in Figure 2. We also modified the guideline explanation in the sidebar to have a brief explanation of all of the detected test smells to ensure that the users are aware of how the test smell classification was performed.

We selected the open-source tool tsDetect [10] to use for finding test smells in the code base. The tool works on Java JUnit projects and has support for JUnit 4 annotations. It also has a published accuracy rating along with classified test smell data. We integrated tsDetect into a service which analyzes a repository stored in GitHub as part of the main analysis performed by BCH. The main advantage of this tool is that it is open source, uses AST-based detection of test smells and supports adding new test smells. In our case, we selected only a subset of the test smells (discussed in subsection 3.2) by restricting the analysis performed on each test file to only these test smells.

3.2 Selection of Test Smells to Evaluate

The tool tsDetect can detect multiple test smells; however, for the purpose of this study we restricted our analysis to the following test smells: conditional test logic; mystery guest; redundant assertion; sensitive equality; verbose test; sleepy test; eager test and resource optimism. The following test smells are not part of the original test smells proposed by van Deursen et al. [3], but are part of the tsDetect tool: conditional test logic, redundant assertion, verbose test; sleepy test. Based on testing done on a manually selected dataset, the tsDetect tool is highly reliable at detecting instances of these test smells [10]. Additionally we evaluated the detection accuracy of the subset of these test smells on two Java projects to confirm these results. The advantage of selecting this specific test smell subset is that each of them does not require viewing the full test source code to understand whether the given test method contains an instance of

[2] `https://bettercodehub.com`

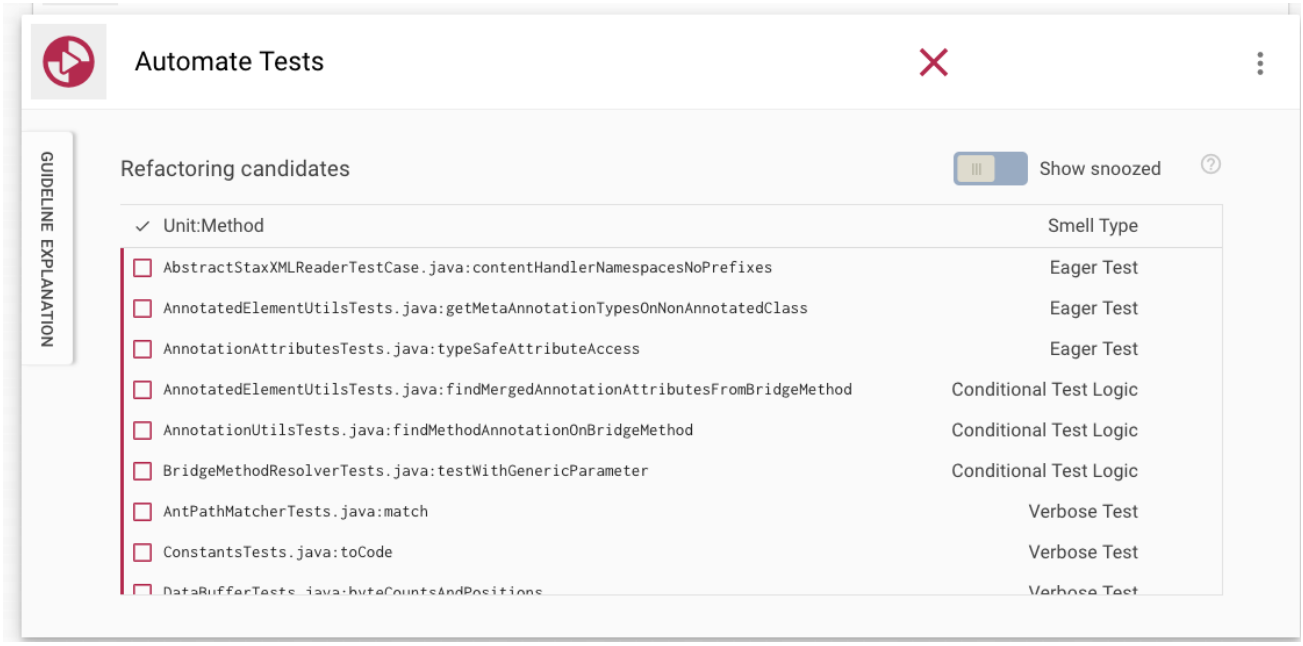

Figure 1: Overview of found test smells in BCH analyzed repository

the test smell or not. Test smells such as general fixture require viewing the entire test code to evaluate, which takes additional time of the developers when compared to analyzing only a single method.

3.3 Survey Design

We created the survey to be integrated into BCH for each test smell based on the following criteria: (1) ability to evaluate if given test smell instance is valid in the project context, (2) option to classify test smell instances as something to fix (refactoring candidate) or as something which will take too much time and effort to fix (technical debt), and (3) at the same time allow developers to rate the subjective importance of the found test smell on the project's maintainability. The first part of the survey helps us answer RQ1, while the second scale rating helps us answer RQ2.

3.4 Participant Selection

Our participants were selected from a pool of Java developers with at least two years of development experience within the host company. Each of them was asked to evaluate parts of the codebase written in Java they were actively working on, while using BCH for the evaluation.

4 Preliminary Results

This preliminary study was done to determine the viability of using BCH as a platform for test smell perception in developers outside of the host organization to gain a larger sample size. We asked 4 developers to use our modified tool and then interviewed them afterwards. The results are summarized in Figures 3 and 4. After a separate interview session with the developers, we were able to obtain additional results not covered by the BCH survey. From the results we can see that the majority of the observed test smells are considered as refactoring candidates. Sensitive Equality shows that the majority of the instances found were false positives, and those which were not considered false positives were ignored. The conditional test logic is evenly split between marking as a refactoring candidate and taking no action.

5 Discussion

The test smell rated as the one which developers decided to ignore the most is conditional testing logic, where the threshold was more than 1 branch statement per test case. During the interviews, developers have repeatedly identified that rewriting the test cases to not use conditions would be either impossible or take too much effort with no additional gain for test maintainability. A similar view was held for eager test, where for certain test cases it was deemed desirable to call multiple production methods in a row before doing an assert. The alternative would be to move the first production call to another method which sets up the test; however, this method might only be used by the one specific test case, negating any benefits of moving to a separate function. Sensitive equality was deemed the most as a false positive or no action item. This was primarily due to limitations of TSDETECT and

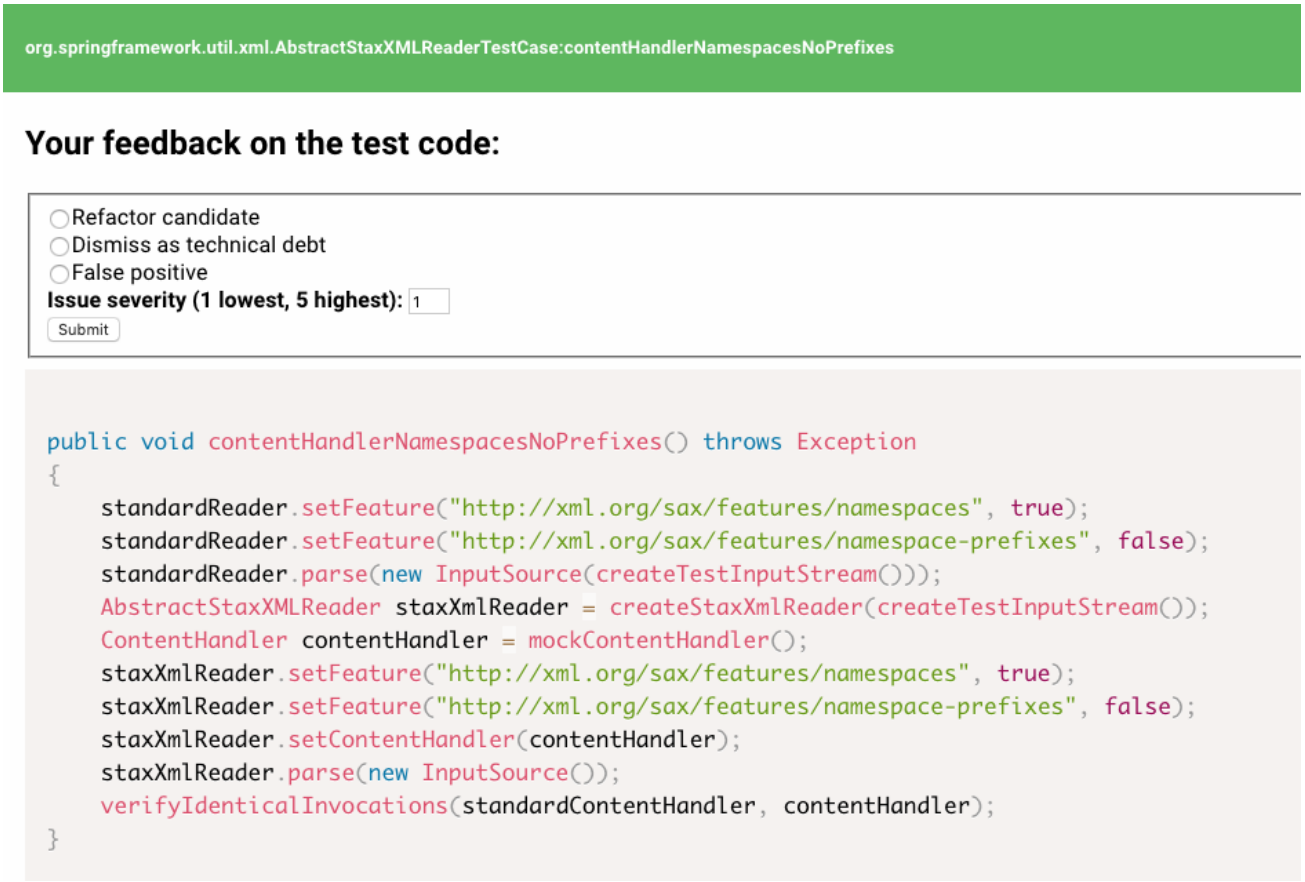

Figure 2: Source code of test with found test smells with user feedback form

some parts of the code working with parsers, where *toString*() was required to evaluate the output. Other test smells were primarily rated as refactoring candidates. These test smells could then be used as a metric for evaluating the quality of the unit tests, as these are the smells developers are willing to remove. From the rankings of test smell presence on maintainability, we can see that the highest impact is sleepy test and redundant assertion. Due to the high false positive rate for sensitive equality, the developers ranked the issue as not severe or hard to avoid.

5.1 RQ1

Based on the results, we can see that developers consider most test smells to have a negative effect on the codebase and should be removed by refactoring. For test smells which are detected reliably, the developers consider the best course of action to take is refactoring the test method to remove the test smell instance. For eager test, the developers expressed that refactoring the test would be difficult and not improve the overall test suite quality. Conditional test logic was the test smell most considered as one to keep (take no action). This can be because the developers consider the test smell hard to remove in the presented cases. Gaining more data about sensitive equality would require using a different tool with a lower false positive rate.

5.2 RQ2

Verbose test was rated always as a refactoring candidate, despite on average being rated as having a low impact. On inspection of the found test smell instances, we found that the tests often contained several blocks with comments and were repeatedly testing the same methods with different values, indicating a co-occurence with lazy test. Sleepy test was rated as having both the highest impact and was always rated as a refactoring candidate, indicating that the removal of dependency on threading in unit tests is perceived as high priority. Resource optimism is considered medium severity and always a refactoring candidate, indicating that this test smell can be easily refactored by adding the extra file existence checks. Nonetheless, if the test fails due to a file not being found, it is easy to detect. Eager test was rated on average as below medium severity and primarily a refactoring candidate, indicating that it is not perceived as a problem. Constructor initialization was rated as

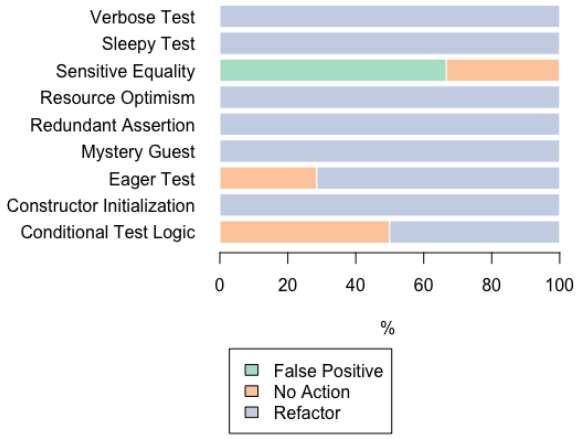

Figure 3: Preliminary results showing the percentage of actions to take for each test smell

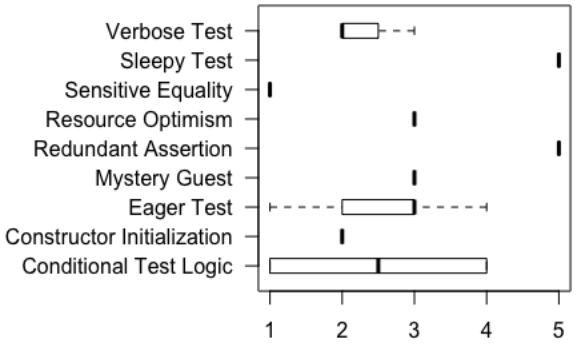

Figure 4: Preliminary results ranking test smell and impact on code maintainability, where 1 is low impact and 5 is highest impact

not severe problem and could be refactored into setup methods.

6 Future Work

The internal preliminary study shows promising results to deploy this modified version of BCH to the general public. This will enable us to analyze multiple projects, both open and closed source from their developers. We plan to determine if there is a relationship between the test smell perception and project experience (in terms of time and commits) and the familiarity with the file where the test smell resides by the commit history.

7 Conclusion

We extended Better Code Hub with test smell detection and ran a preliminary study. In this study, we investigated the developer perception of test smells using a sample of developers from the host company on a codebase they were working on. The results show that the tool can be deployed for the general public and used to gather a larger sample size from the users.

Acknowledgements

We would like to thank the developers at Software Improvement Group for taking part in this study.

References

[1] Gabriele Bavota et al. “Are test smells really harmful? An empirical study”. In: *Empirical Software Engineering* 20 (2014), pp. 1052–1094.

[2] Gabriele Bavota et al. “Are test smells really harmful? An empirical study”. In: *Empirical Software Engineering* 20.4 (Aug. 1, 2015), pp. 1052–1094. ISSN: 1573-7616. DOI: 10.1007/s10664-014-9313-0. URL: https://doi.org/10.1007/s10664-014-9313-0 (visited on 01/15/2019).

[3] Arie van Deursen et al. “Refactoring Test Code”. In: *Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP)*. 2001, pp. 92–95.

[4] Vahid Garousi and Barış Küçük. “Smells in software test code: A survey of knowledge in industry and academia”. In: *Journal of Systems and Software* 138 (Apr. 2018), pp. 52–81. ISSN: 0164-1212. DOI: 10.1016/j.jss.2017.12.013. URL: http://www.sciencedirect.com/science/article/pii/S0164121217303060.

[5] M. Greiler, A. van Deursen, and M. Storey. “Automated Detection of Test Fixture Strategies and Smells”. In: *Verification and Validation 2013 IEEE Sixth International Conference on Software Testing*. Verification and Validation 2013 IEEE Sixth International Conference on Software Testing. Mar. 2013, pp. 322–331. DOI: 10.1109/ICST.2013.45.

[6] Michaela Greiler et al. “Strategies for avoiding text fixture smells during software evolution”. In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. 2013 10th IEEE Working Conference on Mining Software Repositories (MSR 2013). San Francisco, CA, USA: IEEE, May 2013, pp. 387–396. DOI: 10.1109/MSR.2013.6624053. URL: http://ieeexplore.ieee.org/document/6624053/ (visited on 01/28/2019).

[7] Gerard Meszaros. *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.

[8] F. Palomba, A. Zaidman, and A. De Lucia. "Automatic Test Smell Detection Using Information Retrieval Techniques". In: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). Sept. 2018, pp. 311–322. DOI: 10.1109/ICSME.2018.00040.

[9] Fabio Palomba and Andy Zaidman. "Does Refactoring of Test Smells Induce Fixing Flaky Tests?" In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (2017), pp. 1–12.

[10] Anthony Peruma et al. *Software Unit Test Smells*. Software Unit Test Smells. 2018. URL: https://testsmells.github.io/index.html (visited on 04/25/2019).

[11] B. V. Rompaey, B. D. Bois, and S. Demeyer. "Characterizing the Relative Significance of a Test Smell". In: *2006 22nd IEEE International Conference on Software Maintenance*. 2006 22nd IEEE International Conference on Software Maintenance. Sept. 2006, pp. 391–400. DOI: 10.1109/ICSM.2006.18.

[12] B. Van Rompaey et al. "On The Detection of Test Smells: A Metrics-Based Approach for General Fixture and Eager Test". In: *IEEE Transactions on Software Engineering* 33.12 (Dec. 2007), pp. 800–817. ISSN: 0098-5589. DOI: 10.1109/TSE.2007.70745.

[13] Abdus Satter, Nadia Nahar, and Kazi Sakib. "Automatically Identifying Dead Fields in Test Code by Resolving Method Call and Field Dependency". In: (2017), p. 8.

[14] D. Spadini et al. "On the Relation of Test Smells to Software Code Quality". In: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). Sept. 2018, pp. 1–12. DOI: 10.1109/ICSME.2018.00010.

[15] Michele Tufano et al. "An empirical investigation into the nature of test smells". In: *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2016), pp. 4–15.

[16] Hyrum K. Wright, Miryung Kim, and Dewayne E. Perry. "Validity Concerns in Software Engineering Research". In: *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. FoSER '10. event-place: Santa Fe, New Mexico, USA. New York, NY, USA: ACM, 2010, pp. 411–414. ISBN: 978-1-4503-0427-6. DOI: 10.1145/1882362.1882446. URL: http://doi.acm.org/10.1145/1882362.1882446 (visited on 04/30/2019).