Test Refactoring: a Research Agenda

Brent van Bladel
brent.vanbladel@uantwerpen.be

Serge Demeyer
serge.demeyer@uantwerpen.be

University of Antwerp,
Middelheimlaan 1,
2020 Antwerp, Belgium

Abstract

Research on software testing generally focusses on the effectiveness of test suites to detect bugs. The quality of the test code in terms of maintainability remains mostly ignored. However, just like production code, test code can suffer from code smells that imply refactoring opportunities. In this paper, we will summerize the state-of-the-art in the field of test refactoring. We will show that there is a gap in the tool support, and propose future work which will aim to fill this gap.

1 Introduction

Refactoring is "the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure" [Fow09]. If applied correctly, refactoring improves the design of software, makes software easier to understand, helps to find faults, and helps to develop a program faster [Fow09].

In most organizations, the test code is the final "quality gate" for an application, allowing or denying the move from development to release. With this role comes a large responsibility: the success of an application, and possibly the organization, rests on the quality of the software product [Dus02]. Therefore, it is critical that the test code itself is of high quality. Methods, such as code coverage analysis and mutation testing, help developers assess the effectiveness of the tests suite. Yet, there is no metric or method to measure the quality of the test code in terms of readability and maintainability.

Refactoring of the production code can be done with little risk using a test suite, since it provides a safeguard against regressions during software transformation. Tests ensure that the production code preserves its external behaviour pre- and post- refactoring. On the other hand, there is no reliable method to verify if a refactored test suite preserves its external behaviour. Several studies point out the peculiarities of test code refactoring [VDMvdBK01, VDM02, Pip02, Fow09]. However, none of them provided an operative method to guarantee that such refactoring was preserving the behaviour of the test.

The rest of the paper is organized as follows. First we will summerize the related work on test smells and test refactoring, which shows test smells to be an important issue. Then we will go over the existing test refactoring tools, showing there is a gap in the current tool support. Finally, we will propose our future work which aims to fill the gap in existing tool support.

2 Related Work

The term test smell was first introduced by van Deursen et al. in 2001 as a name for any symptom in the test code of a program that possibly indicates a deeper problem. In their paper, they defined a first set of eleven common test smells and a set of specific refactorings which solve those smells [VDMvdBK01]. Meszaros expanded the list of test smells in 2007, making a further distinction between test smells, behaviour smells, and project smells [Mes07]. Greiler et al. defined five new test smells specifically related to test fixtures in 2013 [GvDS13].

Several studies have investigated the impact test smells have on the quality of the code. Van Rompaey et al. performed a case study in 2006 in which they investigated two test smells (*General Fixture* and *Eager Test*). They concluded that all tests which suf-

Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution SATToSE 2017 (sattose.org).
07-09 June 2017, Madrid, Spain.

fer from these smells have a negative effect on the maintainability of the system [VRDBD06]. In 2012, Bavota et al. performed an experiment with master students in which they studied eight test smells (*Mystery Guest*, *General Fixture*, *Eager Test*, *Lazy Test*, *Assertion Roulette*, *Indirect Testing*, *Sensitive Equality*, and *Test Code Duplication*). This study provided the first empirical evidence of the negative impact test smells have on maintainability [BQO+12]. In 2015, they continued their research and performed the experiment with a larger group, containing more students as well as developers from industry. They conclude that test smells represent a potential danger to the maintainability of production code and test suites [BQO+15].

In 2016, Tufano et al. investigated the nature of test smells. They conducted a large-scale empirical study over the commit history of 152 open source projects. They found that test smells affect the project since their creation and that they have a very high survivability. This shows the importance of identifying test smells early, preferably in the IDE before the commit. They also performed a survey with 19 developers which looked into their perception of test smells and design issues. They showed that developers are not able to identify the presence of test smells in their code, nor do developers perceive them as actual design problems. This highlights the importance of investing effort in the development of tools to identify and refactor test smells [TPB+16].

3 Tool Support

There are many tools that can automatically detect code smells, for example the JDeodorant Eclipse plugin and the inFusion tool [FMM+11]. Test smells, however, are very different from code smells and these tools are not able to detect them. Tool support for handling test smells and refactoring test code is limited.

In 2008, Breugelmans et al. presented TestQ, a tool which can statically detect and visualize 12 test smells [BVR08]. TestQ enables developers to quickly identify test smell hot spots, indicating which tests need refactoring. However, the lack of integration in development environments and the overall slow performance make TestQ unlikely to be useful in rapid code-test-refactor cycles [BVR08].

In 2013, Greiler et al. presented a tool which can automatically detect test smells in fixtures [GvDS13]. Their tool, called TestHound, provides reports on test smells and recommendations for refactoring the smelly test code. They performed a case study where developers are asked to use the tool and afterwards are interviewed. They show that developers find that the tool helps them to understand, reflect on and adjust test code. However, their tool is limited to smells related to test fixtures. Furthermore, they only report the occurences of the different fixture-related test smells in the code. They do not give one single metric that represents the overall quality of the test code. During the interviews, one developer said that the different smells should be integrated in one high-level metric: "This would give us an overall assessment, so that if you make some improvements you should see it in the metric." [GvDS13].

Refactoring of the production code can be done with little risk using the test suite as a safeguard. Since there is no safeguard when refactoring test code, there is a need for tool support that can verify if a refactored test suite preserves its behaviour pre- and post-refactoring. Previous research on this topic has been performed by Parsai et al. in 2015 [PMSD15]. They propose the use of mutation testing to verify the test behaviour. However, mutation testing requires the test suite to be ran for each mutant, which can be hundreds of times, making it unlikely to be useful in practice. Furthermore, while mutation testing gives an indication of the test behaviour, it cannot fully guarantee that the behaviour is preserved.

4 Research Plan

As we have shown, there is a lack of tool support when it comes to test refactoring. We plan on creating a tool that will help developers during this process. We present our future work in terms of a research agenda:

Test Smell Detection

- *Objective* - Create a tool that is able to detect test smells. More specifically, the tool should be able to detect all test smells defined by van Deursen, Meszaros, and Greiler [VDMvdBK01, Mes07, GvDS13].
- *Approach* - Breugelmans et al. proposed methods for detecting all the original test smells (defined by van Deursen et al.) [BVR08]. We will use these methods in our tool. For the other test smells (defined by Meszaros and Greiler et al.), we will use a similar approach in order to define detection methods ourselves.
- *Validation* - Verification of correctness will be made using a dataset consisting of a set of real open-source software projects. We can compare the tool with TestHound for fixture related test smells and with TestQ for the other test smells. Smells not covered by either TestHound or TestQ will require manual verification.

Test Quality Metric

- *Objective* - Create a metric that represents the overall quality of the test code. More specifically, the metric should represent the quality of the test code in terms of maintainability rather than effectiveness of the tests. This metric should then be calculated automatically by the tool.

- *Approach* - We will define such a high-level metric in the same way that the inFusion tool defines the overall quality of production code [FMM+11]. The inFusion tool calculates a score that represents the overal quality of the code using a weigthed sum over the traditional code smells. We will define and calculate a similar score for the test code as a weighted sum of test smells.

- *Validation* - Verification of the calculation of the metric can easily be done manually. However, to verify that the metric provides a reasonable representation of the quality of the test code is more challenging. First, we will calculate the metric for the same datasets used for the verification of the smell detection, and check that the score changes appropriately after we apply refactorings or add more test smells. Second, we will compose a dataset of commits from open-source projects that perform a test refactoring, and verify that the score changes appropriately in real world scenarios.

Defining Test behaviour

- *Objective* - Define test behaviour such that developers can verify if the test code is behaviour preserving between pre- and post- refactoring.

- *Approach* - We will use static analysis of the code in order to map all entry and exit points from test code to production code and link them with the corresponding assertions. The production code should be deterministic, and thus the same set of inputs should always result in the same set of outputs. If we can create the input set and expected output set for each assertion, we can use this data as the definition of test behaviour. In case we find that static analysis is not feasible to accomplish this, we will continue using dynamic analysis. We can easily collect this data at run time, and since the test suite is typically run once before and after refactoring, the additional cost is reasonable.

- *Validation* - We will run the algorithm on the dataset of commits used for verifying the test quality metric. We can do an initial check using coverage metrics and mutation testing. When these metrics change pre- and post-refactoring, we know for certain that the test behaviour changed. When these metrics remain constant, we will have to manually verify wether the refactoring is behaviour preserving.

5 Conclusion

We have presented an overview of the research done in the field of test smells and test refactoring. Research has indicated that test smells have a negative impact on maintainability and therefore need to be refactored. We have shown that there is a lack of tool support to aid developers with test refactoring. We plan to create a tool for test refactoring which can detect test code smells, evaluate the test quality, and assure behaviour is preserved after test refactoring. This tool will help developers decide when and where to refactor the test code, as well as help them perform the refactorings correctly, allowing developers to improve their test suite quickly and with confidence.

References

[BQO+12] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and David Binkley. An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 56–65. IEEE, 2012.

[BQO+15] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. Are test smells really harmful? an empirical study. *Empirical Software Engineering*, 20(4):1052–1094, 2015.

[BVR08] Manuel Breugelmans and Bart Van Rompaey. Testq: Exploring structural and maintenance characteristics of unit test suites. In *WASDeTT-1: 1st International Workshop on Advanced Software Development Tools and Techniques*, 2008.

[Dus02] Elfriede Dustin. *Effective Software Testing: 50 Ways to Improve Your Software Testing.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[FMM+11] Francesca Arcelli Fontana, Elia Mariani, Andrea Mornioli, Raul Sormani, and Alberto Tonello. An experience report on using code smells detection tools. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, pages 450–457. IEEE, 2011.

[Fow09] Martin Fowler. *Refactoring: improving the design of existing code*. Pearson Education India, 2009.

[GvDS13] Michaela Greiler, Arie van Deursen, and Margaret-Anne Storey. Automated detection of test fixture strategies and smells. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 322–331. IEEE, 2013.

[Mes07] Gerard Meszaros. *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.

[Pip02] Jens Uwe Pipka. Refactoring in a test first-world. In *Proc. Third Intl Conf. eXtreme Programming and Flexible Processes in Software Eng*, 2002.

[PMSD15] Ali Parsai, Alessandro Murgia, Quinten David Soetens, and Serge Demeyer. Mutation testing as a safety net for test code refactoring. In *Scientific Workshop Proceedings of the XP2015*, page 8. ACM, 2015.

[TPB+16] Michele Tufano, Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. An empirical investigation into the nature of test smells. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 4–15. ACM, 2016.

[VDM02] Arie Van Deursen and Leon Moonen. The video store revisited–thoughts on refactoring and testing. In *Proc. 3rd Intl Conf. eXtreme Programming and Flexible Processes in Software Engineering*, pages 71–76. Citeseer, 2002.

[VDMvdBK01] A Van Deursen, L Moonen, A van den Bergh, and G Kok. Refactoring test code. In *2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001)*, pages 92–95. University of Cagliary, 2001.

[VRDBD06] Bart Van Rompaey, Bart Du Bois, and Serge Demeyer. Characterizing the relative significance of a test smell. In *2006 22nd IEEE International Conference on Software Maintenance*, pages 391–400. IEEE, 2006.