2024 IEEE 17th International Scientific Conference on Informatics (Informatics) | 979-8-3503-8768-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/Informatics62280.2024.10900819

The Impact of Unit Test Quality on Software Development: Controlled Experiment

Patrícia Járosiová and Sergej Chodarev
Department of computers and informatics, Faculty of Electrical Engineering and Informatics
Technical University of Košice
Košice, Slovakia
Email: jarosipatricia@gmail.com, sergej.chodarev@tuke.sk

***Abstract*—This paper presents the results of an empirical study investigating the impact of unit test quality on the following aspects of software development: the development time, the ability to solve all tasks, the perceived software understanding and the presence of adequate feedback. For this purpose, an application was developed with two test suites, one containing high-quality tests and the other low-quality tests. The study was performed on 53 graduate students of computer science, who were randomly divided into 3 groups: students with high-quality tests (Q), students with low-quality tests (L), and students with no tests (N). The goal was to observe the differences between the groups when solving different tasks. Based on the results obtained, it can be said that a significant time difference could be discovered between the groups Q and L, as the Q group completed the individual tasks in a shorter time. In addition, a connection between the presence of quality tests and having adequate feedback has been found. Those who had high-quality tests received appropriate feedback much more often than those who had low-quality tests.**

***Index Terms*—controlled experiment, empirical software engineering, quality of unit tests, unit testing**

I. Introduction

Automatic testing has become a standard part of the software development process. Not only does it help to find errors, but also contributes to better software design and easier development and maintenance [1].

The influence of tests on development productivity is a topic of high research interest [2]. However, not all tests are the same. Low-quality of tests could provide less benefits compared to high-quality test suite, or even hamper the development process.

The purpose of this paper is to find answers to the following question: To what extent does the quality of tests affect the development of a software? We conducted a controlled experiment with three groups of graduate students. They all received a simple React application, but with different test suites:

1) group Q received the high-quality test suite,
2) group L received the low-quality test suite,
3) group N received no tests at all.

We compared times of completing different code maintenance and development tasks, ability to complete the tasks and perceived feedback and code understanding.

II. Quality unit testing

Unit testing can be considered the first level of testing, where the smallest testable parts of the software—called units—are tested independently [3]. As Fowler explained [4], the definition of a unit can be different in every project because it can be strongly dependent on the given architecture. Generally, the definition of the units should correspond more to the units of behavior and not to structure of the code.

There are multiple guidelines and best-practices for writing unit-tests. For example, Hunt and Thomas describe the following characteristics of a well-written test in their book [5]:

- Automated—calling the tests as well as checking the results should be automatic.
- Thorough—all key paths and scenarios that can lead to problems need to be tested.
- Repeatable—the tests must always pass and produce the same results every time they are run.
- Professional—they must be written and maintained according to professional standards.
- Independent—they need to be kept independent of each other and also of the environment.

There are also multiple books describing patterns of unit-tests—well-established solutions to common problems [6], [7]. In addition to the patterns, it is also possible to discuss different types of antipatterns, also called test smells, that should be avoided. Test smells are recurring problems that can be mitigated using proper application of test patterns. The concept of test smells is based on the bad code smells [8]. While test smells and code smells in general are not errors in all cases, they represent characteristics of the code that indicate possible problems.

The following can be classified among the most famous and most frequently occurring test smells [6], [9]:

- Assertion Roulette—a test with multiple assertions, each of which checks a different condition.
- Duplicate assert—a test that checks the same thing with the same parameters multiple times.
- Eager test—a test that contains multiple calls to multiple different production methods.
- Empty test—a test without executable statements.
- Sleepy test—a test in which the execution of the statements is paused for a certain period of time.

- Unknown test—a test without any assertion.
- Fragile test—a test that fails when unrelated production code changes.
- Erratic test—a test that sometimes passes but sometimes fails depending on when they are run and who is running them.

It can therefore be said that the quality of the tests can be maintained by following several methods and recommendations.

III. Related Work

Some projects already tried to research general impact of testing to the code quality. For example, Gren and Antinyan performed a survey assess the correlation between practitioners' perception of code quality and unit testing practices [10]. However, they didn't find strong correlation.

Several studies analyzed the impact of test-driven development on quality of tests [11] and overal quality of code and productivity [2]. While most of the studies found that code quality is improved by test-driven development compared to test-last development, the productivity of developers may, in some cases, decrease.

Bavorta et al. performed a series of studies [12], [13] where they assess prevalence and impact of bad code smells. They have found that test smells are very common in analyzed open-source and industrial projects. A part of their research were experiments in both academic and industrial settings. They performed a within-subject study where they compared code comprehension and maintenance tasks with presence and absence of test code smells. The tasks required to answer questions about code, but not the actual modification of it. As the result they have found that test smells negatively impact programmer comprehension.

A study by Spadini et al. [14] also demonstrates that test code smells decrease quality of both tests code and production code tested. The results are based on the release history data from 10 open-source Java projects and analyzed change-proneness and defect-proneness of the code.

IV. Method

A web application was prepared for the experiment, which was created specifically for this purpose. Furthermore, 2 test suites were created, one of which contains high-quality tests, while the other contains low-quality tests.

The study was designed as a controlled experiment complemented by a questionnaire. It has a between-groups design, meaning that each participant experiences only one condition, either working with high-quality tests (Q), low-quality tests (L), or no tests (N).

Three tasks were created for the experiment, each of them was designed to be easy to solve for all participants. The goal was to simulate various situations that usually occur during software development, namely: changing the functionality of an existing element (task 1), changing the implementation of an existing functionality (task 2), creating a new element (task 3).

A. Hypotheses

We formulated 9 null hypotheses (H_0) and their corresponding alternative hypotheses (H_a).

First 6 hypotheses deal with the impact of test quality on development time in each of the development situations (changing functionality, changing implementation, adding a new functionality). In all cases we compare group with high-quality tests with either low-quality (L) or no-tests (N) groups. We expected that high-quality tests would lead to shorter development time compared to low-quality, and that there would be some significant difference in time compared to no-tests group.

Therefore, the following hypotheses were formulated:

1) $\mathbf{HL1_0}$: The average time required to solve a task in which a functionality of an element is changed, with the presence of quality unit tests is greater than or the same as with the presence of low-quality unit tests.
   $\mathbf{HL1_a}$: The average time required to solve a task in which a functionality of an element is changed, with the presence of quality unit tests is less than with the presence of low-quality unit tests.
2) $\mathbf{HN1_0}$: There is no difference between the Q and N groups in the time measured for solving a task in which the functionality of an element is changed.
   $\mathbf{HN1_a}$: There is a difference between the Q and N groups in the time measured for solving a task in which the functionality of an element changes.
3) $\mathbf{HL2_0}$: The average time required to solve a task, in which the implementation of a functionality is changed, with the presence of quality unit tests, is greater than or the same as with the presence of low-quality unit tests.
   $\mathbf{HL2_a}$: The average time required to solve a task, in which the implementation of a functionality is changed, with the presence of quality unit tests, is less than with the presence of low-quality unit tests.
4) $\mathbf{HN2_0}$: There is no difference between groups Q and N in the time measured for solving a task, in which the implementation of a functionality is changed.
   $\mathbf{HN2_a}$: There is a difference between groups Q and N in the time measured for solving a task, in which the implementation of a functionality is changed.
5) $\mathbf{HL3_0}$: The average time required to solve a task in which new elements are added to a component with the presence of quality unit tests is equal to or greater than with the presence of low-quality unit tests.
   $\mathbf{HL3_a}$: The average time required to solve a task in which new elements are added to a component with the presence of quality unit tests is less than with the presence of low-quality unit tests.
6) $\mathbf{HN3_0}$: There is no difference between the Q and N groups in the time taken to solve a task in which new elements are added to the component.
   $\mathbf{HN3_a}$: There is a difference between the Q and N groups in the time taken to solve a task where new elements are added to the component.

Additional hypotheses are related to whatever participants in different groups would able to *solve* all the tasks (HS), and what would be their subjective perception of received *feedback* (HF) and *understanding* of the software source code (HU).

1) $\mathbf{HS_0}$: There is no relationship between the presence of quality unit tests and the ability to solve all tasks.
   $\mathbf{HS_a}$: There is a relationship between the presence of quality unit tests and the ability to solve all tasks.
2) $\mathbf{HF_0}$: There is no relationship between the presence of quality unit tests and the adequacy of perceived feedback.
   $\mathbf{HF_a}$: There is a relationship between the presence of quality unit tests and the adequacy of perceived feedback.
3) $\mathbf{HU_0}$: There is no relationship between the presence of quality unit tests and the perceived understanding of the software.
   $\mathbf{HU_a}$: There is a relationship between the presence of quality unit tests and the perceived understanding of the software.

B. Project and tests

The application was a simple personal organizer allowing one to manage tasks, calendar events, links, and running timers. It was implemented in JavaScript programming language using the popular React framework[1].

The tests for the experiment were written based on the information briefly summarized in Section II. However, since these are quite standard practices, in the case of specific uncertain situations, the comments and advice of experts also played an important role in achieving a higher level of test quality. Low-quality tests, on the other hand, contained typical problems characterized by the test smells.

Overall there were 71 test cases in the high-quality test suite, while the low-quality test suite contained 64 test cases. The tests were written using the Jest testing framework[2] and checked mainly the functionality of the application user interface and the interaction with the database[3].

The quality of the test suites was also verified by generating code coverage reports, from which the following results were obtained: 100% line coverage, 99.49% statement coverage, and 97.67% branch coverage. Based on the generated report, it can be said that the results achieve optimal code coverage, which increases confidence in the test suite.

C. Form of data collection

Two forms of data were collected: screen recordings of participants performing the tasks and the results of the completed questionnaires. For each participant, the screen recording was evaluated and the time to complete the tasks was measured, as well as the number of tasks they were able to successfully solve. The obtained data were then organized and used for later statistical analysis.

[1]https://react.dev/
[2]https://jestjs.io/
[3]Source code is available at https://doi.org/10.17605/OSF.IO/862YT

For the purpose of development time analysis, only the measured times of successfully solved tasks were compared.

D. Participants

The participants of this study were graduate students of Computer Science at the Technical University of Košice studying the Metaprogramming course. A total of 53 people participated in the experiment, of which 20 people belonged to the Q group, 15 people to the L group and 18 people to the N group. All of them agreed to participate in the experiment and to have screen recording.

E. Setup and procedure

The experiment took place in the main building of the Technical University of Košice, in a classroom, where 24 computers are available. On these computers, the following programs and files were prepared in advance for each participant:

- ScreenRec[4] for screen recording,
- PowerPoint presentation with instructions for performing the experiment, with a short instructional video of the application and also with the tasks that the participants needed to solve,
- Visual Studio Code[5] with opened and prepared projects.

The instructions for performing the experiment were given at the beginning of the experiment but were also available in written form in the documentation. The projects were prestarted on the machines. When participants were ready to start solving the tasks, they had to first start a screen recording and then leave a comment about when they started solving (and then when they finished) a particular task to help us determine the time it took to complete that task. After finishing, they turned off the recording and filled out the questionnaire.

We have chosen the Visual Studio Code as a development environment to ensure that all participants have the same conditions for solving the tasks.

F. Used statistical methods

The level of significance in all cases presented in this paper uses the most recommended value, which is α = 0.05.

In the case of this experiment, 3 different methods [15] were used, namely: two-sample t-test, Mann-Whitney U test, Chi-Square test.

One-tailed two-sample t-test is used for hypotheses $HL1_0$, $HL2_0$ and $HL3_0$. The suitability of this test for these hypotheses is verified by the fact that all its following rules are fulfilled for all the mentioned hypotheses: the samples are independent, the variables are normally distributed (verified with the Anderson Darling test), variables are metric, variances within groups are equal/similar (verified with Levene's test).

Mann-Whitney U test is used for hypotheses $HN1_0$, $HN2_0$ and $HN3_0$. This test is suitable for these hypotheses because it is a non-parametric alternative test to the two-sample t-test. An alternative solution is needed because in

[4]https://screenrec.com/
[5]https://code.visualstudio.com/

these cases the data are not normally distributed and the variances are not equal, which means that the assumptions of the t-test are not met.

Chi-Square test is used for hypotheses HS_0, HF_0 and HU_0. The suitability of this test for these hypotheses is verified by the fact that all of its following rules are fulfilled for all the mentioned hypotheses: they have two categorical variables and two or more categories for each variable, observations are independent, the expected frequency for each cell is at least 1, and the expected frequency is at least 5 for 80% of the cells.

G. Questionnaire

Each group received a different questionnaire, but they all contained questions regarding the participants' experience with JavaScript and React, as well as the comprehensibility of the tasks.

In addition to the above questions, the following questions were also present in the questionnaire for groups Q and L:

- Q1: Were the tests helpful in solving the tasks?
- Q2: Do you think you could solve the tasks faster without tests?
- Q3: Did the tests help you make sure you solved the problems correctly?
- Q4: Did the tests help you make sure that the changes you made to the code didn't cause other components to fail?
- Q5: Did the tests help to document the expected behavior of a specific component and thereby better understand the functioning of the component?
- Q6: Did it take you a long time to understand what the tests of the given component are for or what functionality they test?
- Q7: Did it often happen that one of the tests failed when solving the tasks?
- Q8: Was there a time when it wasn't clear why a particular test failed?

As for group N, since they did not have any tests, the following questions were included in their questionnaire:

- Q1: Do you think the presence of unit tests would speed up the solving of tasks?
- Q2: Do you think the presence of unit tests would make it easier to understand how the application works?
- Q3: Do you think that the presence of unit tests would provide more reliable feedback that there were no errors in the operation of other components or elements while solving the tasks?

Participants could choose from the following 5 answers to these questions: definitely yes, rather yes, I don't know, rather not, definitely not.

V. Results

A. Questionnaire results

It can be said that 92% of the participants have at least basic experience with JavaScipt and 74% of them have some experience with React, so the chosen language and framework should not be problem in solving the tasks. Also, the tasks were completely understandable for 96% of the participants. Based on the obtained percentages, it can be said with certainty that the tasks are sufficiently understandable and thus do not negatively affect the obtained results.

The answers of the members of groups Q and L can be seen in Figure 1. It can be said that these 2 groups have quite opposite views. For example, while 75% of the Q group think they received adequate feedback, only 25% of the L group feel the same way. There is a similarly big difference in the answers to questions Q3 and Q5.

The answers of the members of group N can be seen in Figure 2. The given figure clearly shows that the majority of participants in group N believe that the tests would have a more positive effect on solving the tasks. The negative impact of their absence was mainly manifested in the lack of feedback.

B. Impact of test quality on development time

1) Changing functionality: The data collected from the solution of the task in which the functionality of an element is changed are displayed graphically in Figure 3. It can be seen that the average time measured for the Q group is the smallest.

Conclusion for HL1: Since we obtained a p-value of 0.001, which is less than the 0.05 significance level, the alternative hypothesis can be **accepted**. Therefore it is possible to come to the conclusion that the average time to solve a task in which the functionality of an element is changed, with the presence of quality unit tests, is less than with the presence of low-quality unit tests.

Conclusion for HN1: Since the obtained p-value of 0.196 is above the 0.05 significance level, there is **not enough evidence** to reject the null hypothesis. There is no significant difference between the Q and N groups in the time measured for solving a task in which the functionality of an element is changed.

2) Changing implementation: The data collected from the solution of the task in which the implementation of a functionality is changed are displayed graphically in Figure 4. It can be seen that the average time measured for the Q group is the least.

Conclusion for HL2: Since we obtained a p-value of 0.000004, which is less than the 0.05 significance level, the alternative hypothesis can be **accepted**. Therefore, it is possible to come to the conclusion that the average time to solve a task in which the implementation of a functionality is changed, with the presence of quality unit tests, is less than with the presence of low-quality unit tests.

Conclusion for HN2: Because the obtained p-value of 0.765 is above the 0.05 significance level, there is **not enough evidence** to reject the null hypothesis. There is no significant difference between groups Q and N in the time measured for solving a task, in which the implementation of a functionality is changed.

3) Adding functionality: The data collected from the solution of the task in which new elements are added to a component are displayed graphically in Figure 5. It can be

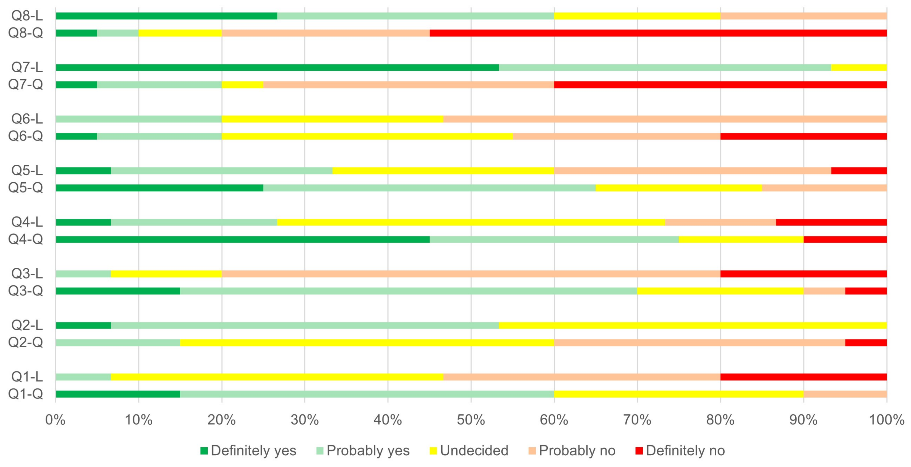

Fig. 1. Questionnaire results for groups Q and L

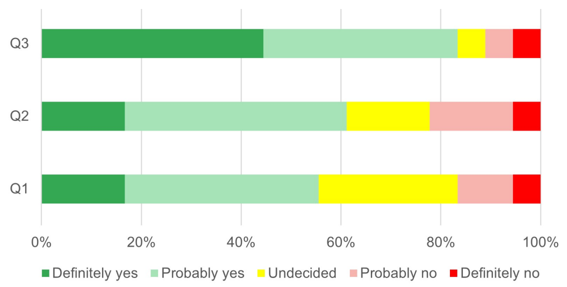

Fig. 2. Questionnaire results for group N

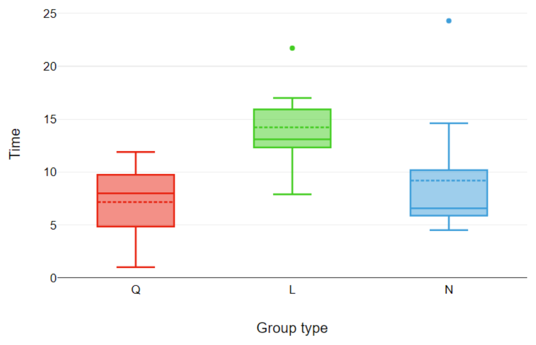

Fig. 4. Boxplot for the results of T2

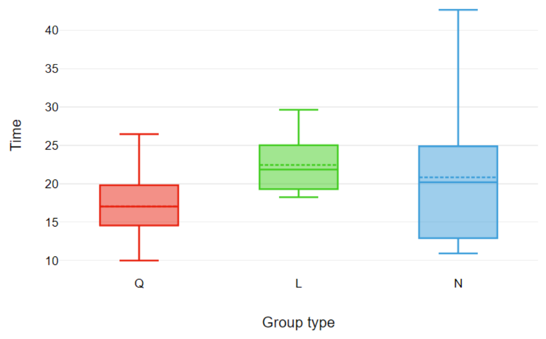

Fig. 3. Boxplot for the results of T1

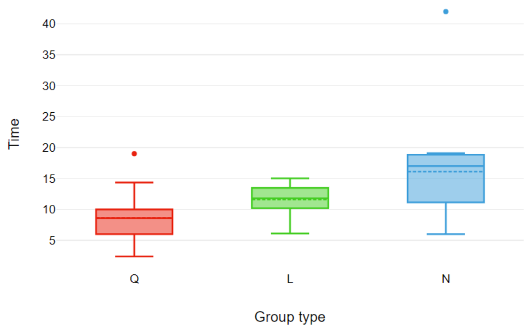

Fig. 5. Boxplot for the results of T3

seen that the average time measured for the Q group is the least.

Conclusion for HL3: Since we obtained a p-value of 0.03, which is less than the 0.05 significance level, the alternative hypothesis can be **accepted**. Therefore, it is possible to come to the conclusion that the average time required to solve a task in which new elements are added to a component with the presence of quality unit tests is less than with the presence of low-quality unit test.

Conclusion for HN3: Because the obtained p-value of 0.001 is below the 0.05 significance level, it can be said that there is enough evidence to reject the null hypothesis and the alternative hypothesis can be **accepted**. There is a significant difference between the Q and N groups in the time taken to solve a task where new elements are added to the component.

C. Impact of test quality on the ability to solve all tasks

The data were collected from screen recordings.

Conclusion for HS: Since the obtained p-value of 0.4 is above the 0.05 significance level, there is **insufficient evidence** to conclude that there is a relationship between the presence of quality unit tests and the ability to solve all tasks, so the null hypothesis is retained.

D. The impact of test quality on feedback

The data were collected through a questionnaire, specifically answers to question number Q4 from Section IV-G.

Conclusion for HF: Since the obtained p-value of 0.01 is below the 0.05 significance level, the null hypothesis can be rejected and the alternative hypothesis **accepted**. There is a relationship between the presence of quality unit tests and the adequacy of perceived feedback. In addition, using adjusted residuals and Bonferroni correction, it is possible to say that people from group Q had significantly more adequate feedback, while people from N had significantly less adequate feedback than expected.

E. The impact of test quality on perceived code understanding

The data were collected through a questionnaire, specifically answers to question number Q5 from Section IV-G.

Conclusion for HU: Since the obtained p-value of 0.141 is above the 0.05 significance level, there is **not enough evidence** to reject the null hypothesis. Based on the data collected, there is no relationship between the presence of quality unit tests and perceived understanding of the software.

VI. Threats to validity

In terms of internal validity, the fact that the screens were recorded can cause unreliable results because the feeling of being constantly observed can change the natural behavior of the participants. Still, it was an important aspect because it helped to ensure that the experiment was performed as expected.

Another threat is the fact that the participants were aware to which group thy belonged to, which could have influenced their self-evaluation by the questionnaire.

Regarding external validity, the tasks and the whole project used in the experiment were simple, which allowed the participants to complete the tasks without major difficulties, on the other hand, a more complex project could better clarify the differences between the groups.

The results could also be influenced by the domain and technologies used in the project. It was a front-end web application consisting mostly of user interaction with minimal business logic. The results, therefore, may not be generalizable to different kinds of projects, for example, to a libraries or back-end of web applications.

VII. Conclusion

The experiment results indicate that high-quality unit tests significantly reduce development time compared to low-quality tests when changing functionality, modifying implementation, and adding new functionality. However, compared to the no-tests group, there was a significant difference only when adding new functionality, but no significant difference in development time for changing functionality and implementation tasks. Additionally, high-quality tests did not significantly impact the ability to solve all tasks.

Based on the self-evaluation questionnaire, the presence of high-quality tests was associated with more adequate feedback but did not significantly impact the perceived understanding of the software code.

The results we obtained indicate that although the quality of tests does not affect software development in all areas, they play an important role in aspects such as development time and adequate feedback. Future research could build on this study by exploring more complex projects or different domains to better generalize the findings and mitigate some of the current study's validity threats.

The source code of the project used in the experiment and the results are available to the public at https://doi.org/10.17605/OSF.IO/862YT (accessed on 13 October 2024).

Acknowledgment

This work was supported by project VEGA 1/0630/22: Lowering Programmers' Cognitive Load Using Context-Dependent Dialogs.

References

[1] P. Kua, *Unit testing*, 1st ed., Oracle Australian Development Centre - Oracle Corporation, Brisbane, Australia, 3 2015.

[2] W. Bissi, A. G. Serra Seca Neto, and M. C. F. P. Emer, "The effects of test driven development on internal quality, external quality and productivity: A systematic review," *Information and Software Technology*, vol. 74, pp. 45–54, Jun. 2016.

[3] P. Mahajan, "Different types of testing in software testing," *International Research Journal of Engineering and Technology*, vol. 3, no. 4, 4 2016.

[4] M. Fowler. (2014, 5) Unit test. [Online]. Available: https://martinfowler.com/bliki/UnitTest.html

[5] A. Hunt and D. Thomas, *Pragmatic Unit Testing in Java with JUnit*, 1st ed. The Pragmatic Programmers LLC, 2003.

[6] G. Meszaros, *xUnit Test Patterns: Refactoring Test Code*, 1st ed. Boston, United States: Pearson Education, 5 2007.

[7] L. Koskela, *Effective unit testing: a guide for Java developers*. Manning Publications Company, 2013.

[8] M. Fowler and K. Beck, *Refactoring: Improving the Design of Existing Code*, 2nd ed. Boston: Addison-Wesley, 2019.

[9] A. Peruma, K. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni, and F. Palomba, "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*. Virtual Event USA: ACM, Nov. 2020, pp. 1650–1654.

[10] L. Gren and V. Antinyan, "On the relation between unit testing and code quality," in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. Vienna, Austria: IEEE, Aug. 2017, pp. 52–56.

[11] A. Tosun, M. Ahmed, B. Turhan, and N. Juristo, "On the effectiveness of unit tests in test-driven development," in *Proceedings of the 2018 International Conference on Software and System Process*. Gothenburg Sweden: ACM, May 2018, pp. 113–122.

[12] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley, "An empirical analysis of the distribution of unit test smells and their impact on software maintenance," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. Trento, Italy: IEEE, Sep. 2012, pp. 56–65.

[13] ——, "Are test smells really harmful? An empirical study," *Empirical Software Engineering*, vol. 20, no. 4, pp. 1052–1094, Aug. 2015.

[14] D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, and A. Bacchelli, "On the relation of test smells to software code quality," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Madrid: IEEE, Sep. 2018, pp. 1–12.

[15] Datatab - online statistics calculator. https://datatab.net. Graz, Austria.