Reflecting the Adoption of Software Testing Research in Open-Source Projects

Fabian Trautsch
Institute of Computer Science
Georg-August-University Göttingen
Göttingen, Germany
trautsch@cs.uni-goettingen.de

Abstract—**In the recent years, a lot of research has been done in the field of software testing. But, there exist few empirical studies which analyze, if results of software testing research are actually practiced in real software projects, why they are (not) practiced, and how this influences the quality of the project. Our proposed research project tries to close this gap by analyzing open-source software projects. We focus our work on a concept, which is well accepted and known in our community for a longer period of time: test levels.**

Hence, we propose a two step approach to tackle the problem. First, we want to determine if the concept of a unit is still up-to-date and propose alternatives otherwise. Furthermore, we aim to investigate, why developers think that the concept of a unit is (not) current. In the second step we intend to check, based on the unit definition determined in the first step, how many tests on the different levels exist for the investigated projects. Additionally, based on the results, we want to examine, why developers are (not) developing tests for a certain test level and how this influences the software quality of the project.

Our initial study showed, that very few projects have unit tests, using the unit definition of the IEEE and ISTQB. Furthermore, it revealed that developers intend to write unit tests, but they fail to do so.

I. Problem Statement and Research Hypothesis

A lot of research has been done in the field of software testing. But, very few empirical studies (like [1]) are performed to analyze, if research results are actually practiced in software projects. This information would also help to determine, if taught lessons are actually practiced. Additionally, based on this data, the influence of (not) practicing these lessons on software quality could be investigated. The results of this kind of research could help to, e.g., improve the training of future developers or give recommendations for future software development.

We have designed three different research hypotheses to stir our research:

- **Hypothesis 1**: Open-source developers do not apply concepts, that are well established in software quality research.
- **Hypothesis 2**: They are not applied, because developers think they are hard to implement in real software projects.
- **Hypothesis 3**: The software quality decreases by not applying these concepts.

Therefore, we want to check if testing concepts from research are applied in real open-source software projects. If they are not applied, we need to check the reasons for this. Furthermore, we also want to check how the software quality is influenced by (not) applying these concepts.

We focus our research on the concept of test levels, which is well accepted and discussed in the software engineering community since at least 1986 [2]. It is important that the concept is in the community for a longer period of time, as it needs time to propagate through the developer community. Furthermore, the concept of having tests that focus on different levels of the system is used in a variety of software development models, like the spiral model [3]. Additionally, much research is focusing on tests on the different levels like [4].

The first step is to check if developers apply the concept of a unit, like it is defined by, e.g., the Institute of Electrical and Electronics Engineers (IEEE) or the International Software Testing Qualification Board (ISTQB). This is the requirement for investigating, if the concept of unit testing, integration testing, and system testing is applied by the developers, as the definition of unit is essential for separating tests into different levels. The results will highlight, if the concept of test levels is applied in software projects. For each test level and the definition of unit, we need to investigate why the concepts are (not) applied. Additionally, we need to analyze, for each test level, if the application of the concepts has an impact on software quality.

To the best of our knowledge this problem has not yet been addressed. There exist single studies, that investigate how developers use certain concepts or practices (e.g., [1]), but none of them focus on test levels and the impact of (not) following these concepts.

II. Research Approach and Expected Contributions

Our research approach can be divided into two different steps: 1) unit definition and 2) test level detection. For each step, we need to gather data to find support for or against our hypothesis. In the first step, we need to determine if the available definitions of a unit (e.g., from the IEEE or ISTQB) are usable and reflected in the practice. Therefore, we will use our SmartSHARK [5] platform to collect data to investigate, how many units are in the examined projects based on available definitions and compare this with self-made definitions. One approach to create these definitions, is to look at the commit behavior of developers and apply an association rule mining algorithm to find files which are often changed together [6]. This would indicate, which files might form a unit. Another approach that we want to test is to use techniques from the field of social network analysis to detect communities within a software project [7], which might give hints on the separation into units. But this is in an early stage of research. This step is planned to be evaluated by a user study, where we ask developers to look at our results and choose the one, which has detected the units of the system best (i.e., IEEE, ISTQB, or own definition of unit) and why the others do not fit.

After we have found a good working definition of the concept of an unit, we will use our SmartSHARK platform to mine data from software projects. This data is used to categorize tests of the projects, using the unit definition found in step 1, into different test levels. Hence, we will analyze how many unit tests, integration tests, or system tests the project really possess and also how the number of tests in the different levels have evolved over time during the project. Furthermore, we want to find support for or against hypothesis 3. Hence, we plan to correlate the results of step 2 with, e.g., the software quality of the project measured by the number of issues over time. Additionally, we aim to investigate the fault detecting capabilities of tests on the different levels. The results of this investigation can give us hints regarding the importance of unit tests, integration tests, and system tests in respect to software quality.

The evaluation of our results will be achieved by performing another user study. In this study, we want to confront the developers with our results. This way, we can determine why developers have developed a test for a specific test level and compare it with their intention. The intention of the developers of writing a test for a certain test level can be mined from the project data (e.g., commit message or separation of tests of the projects into specific folders).

Our expected contributions are as follows:

- an update of the current definition of a unit
- an in-depth analysis of the usage of unit tests, integration tests, and system tests in open-source software projects
- an in-depth analysis of the reasons for (not) unit tests, integration tests, and/or system tests in open-source software projects
- an in-depth analysis of the impact of performing unit testing, integration testing, system testing

III. Dissemination Plan

Our initial study was already accepted on the ICST 2017 [8]. Furthermore, we plan to publish the results of each of the steps explained in Section IIResearch Approach and Expected Contributionssection.2, as the results will extend the existing body of knowledge in the field of software testing. Additionally, other researchers might profit from the results, as it is interesting to see, if research results are actually applied in practice and what the outcome is if they are not practiced.

Furthermore, we plan to publish all our tools (as open-source) and data sets, and make them publicly available in a reusable manner, such that all studies can be replicated. This can also help new researchers in the field to start with their research, as they can reuse, e.g., our data collection tools.

IV. State of the Research

We already had a first look at open-source Python projects [8] to examine, if they use unit tests at all. For this, we used the definitions of the IEEE and the ISTQB for an unit. We analyzed data from over 70K revisions of 10 different python projects and calculated the actual number of unit tests (based on the definitions). Furthermore, we compared this number with the expected number of unit tests, as inferred from the intentions of the developers. Additionally, we investigated the mocking behavior of the developers and its influence on the calculated numbers together with the evolution of the number of unit tests.

Our paper presents five results: 1) developers believe that they develop more unit tests than they actually do, 2) most of the examined projects have a very small amount of unit tests, 3) developers use mocking frameworks, but the usage does not influence the number of unit tests, 4) we detected four different patters of the evolution of the number of unit tests, and 5) the used unit test definition influences the results.

Currently, we are working on the improvement of our data collection tools and want to publish them in a re-usable manner. First tools are already published on GitHub [9]. Furthermore, we plan to redo our study with projects that use another programming language (e.g., Java). Afterwards, the next step is, as explained in Section IIResearch Approach and Expected Contributionssection.2, the creation and evaluation of new definitions of the term unit.

Acknowledgments

I would like to thank my supervisor Jens Grabowski for his support and fruitful comments.

References

[1] P. Runeson, "A survey of unit testing practices," *IEEE software*, vol. 23, no. 4, pp. 22–29, 2006.

[2] P. Rook, "Controlling software projects," *Software Engineering Journal*, vol. 1, no. 1, p. 7, 1986.

[3] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[4] G. Meszaros, *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.

[5] F. Trautsch, S. Herbold, P. Makedonski, and J. Grabowski, "Adressing problems with external validity of repository mining studies through a smart data platform," in *Proceedings of the 13th International Workshop on Mining Software Repositories*. ACM, 2016, pp. 97–108.

[6] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004.

[7] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[8] F. Trautsch and J. Grabowski, “Are there any unit tests? an empirical study on unit testing in open source python projects,” in *Proceedings of the International Conference on Software Testing, Verification and Validation (ICST 2017).* IEEE, 2017, to appear.

[9] F. Trautsch and S. Herbold, “SmartSHARK GitHub Page,” https://github.com/smartshark, [accessed 18-January-2017].