Test Case Quality in Test Driven Development: A Study Design and a Pilot Experiment

Adnan Čaušević, Daniel Sundmark and Sasikumar Punnekkat
Mälardalen University, Sweden
firstname.lastname@mdh.se

Abstract*—*Background: **Test driven development, as a side-effect of developing software, will produce a set of accompanied test cases which can protect implemented features during code refactoring. However, recent research results point out that successful adoption of test driven development might be limited by the testing skills of developers using it.**

Aim: **Main goal of this paper is to investigate if there is a difference between the quality of test cases created while using test-first and test-last approaches. Additional goal of this paper is to measure the code quality produced using test-first and test-last approaches.**

Method: **A pilot study was conducted during the master level course on Software Verification & Validation at Mälardalen University. Students were working individually on the problem implementation by being randomly assigned to a test-first or a test-last (control) group. Source code and test cases created by each participant during the study, as well as their answers on a survey questionnaire after the study, were collected and analysed. The quality of the test cases is analysed from three perspectives: (i) code coverage, (ii) mutation score and (iii) the total number of failing assertions.**

Results: **The total number of test cases with failing assertions (test cases revealing an error in the code) was nearly the same for both test-first and test-last groups. This can be interpreted as "test cases created by test-first developers were as good as (or as bad as) test cases created by test-last developers". On the contrary, solutions created by test-first developers had, on average, 27% less failing assertions when compared to solutions created by the test-last group.**

Conclusions: **Though the study provided some interesting observations, it needs to be conducted as a fully controlled experiment with a higher number of participants in order to validate statistical significance of the presented results.**

***Index Terms*—software testing; test case efficiency; test driven development; controlled experiment;**

I. Introduction

Empirical investigations of Test Driven Development practice (TDD) are being constantly performed from the moment TDD was introduced [1] as part of the eXtreme Programming (XP) methodology. Those experiments are focusing, to a high extent, on evaluating effects of TDD on the quality of code. However, a very small number of experiments is investigating the quality of the produced tests while using TDD. The quality of those tests is important because developers who use TDD have to create automated unit tests *before* the actual code is written (in literature TDD is sometimes referred to as *test-first* approach). Those tests are in the form of assertions and in TDD they define code requirements. This way, developers evolve the systems through cycles of test, development and refactoring.

TDD is gaining high attention in industry. This was the outcome of an industrial survey [2] conducted for the purpose of identifying differences between the preferred and the actual level of usage for several test-related practices. A main finding of this study could be interpreted as: "Respondents would like to use TDD to a significantly higher extent than they actually do currently". This interest could be due to the success stories of early adopters as well as academic research results which are pointing to code quality and productivity improvements when TDD is used ([3] [4] [5] [6] [7]).

What remains unclear are reasons for still avoiding full scale adoption of TDD in the industrial context. By performing a systematic literature review, seven potentially limiting factors of full TDD adoption were identified from the existing academic investigations [8]. Developers inability to write efficient and effective automated test cases is considered to be one of the limiting factors. Goals of this paper are formulated and defined in a way to validate the significance of such a limiting factor.

A. Problem Statement

Test driven development, as a side-effect of developing software, will produce a set of accompanied test cases which can protect implemented features during code refactoring. However, recent research results are pointing out that successful adoption of test driven development might be limited by the testing skills of developers using it. We are interested in comparing testing efforts of a TDD developers and traditional, test-last, developers. This comparison could provide us with further insight if TDD adoption is indeed limited by the developers testing skills.

B. Research Objective

Research objective of this study can be expressed as follows:

> To compare *efficiency* and *effectiveness* of the *testing effort* produced by *test-first* and *test-last* developers.

C. Context

To perform comparison with respect to the above objective, an experiment was organised with master students enrolled in the Software Verification and Validation course at Mälardalen University during the autumn semester in 2011.

D. Paper Outline

This paper is structured according to the reporting guidelines provided in [9] with some minor deviations. In section II

we present the related research works followed by the experimental design in section III. Section IV presents the details of execution of our experiment. The treatment and analysis of the collected data are given in section V. In section VI, we present statistical inferences followed by conclusions and future research planned in section VII.

II. Related Work

In a recent systematic literature review [8], 48 empirical studies were identified with the focus on investigating effects of TDD. Most of the studies had TDD as a primary focus of investigation, but in some cases effects of TDD were investigated together with some other practice, e.g. pair-programming. Most of the studies were investigating effects of TDD with respect to: (i) the internal or the external code quality improvements, (ii) performance improvements or (iii) a general perception of using TDD.

There was only one study identified, which had focus on investigating quality attributes of test cases created using test-first approach. Madeyski [10] investigated how test-first programming can impact branch coverage and mutation score indicators. In his experiment, 22 students were divided in test-first and test-last groups with the task of developing a web based paper submission system. Experiment was performed during 9 weeks period. Results of this experiment point out to no statistically significant difference between the test-first and the test-last groups with respect to branch coverage and mutation score indicators.

Our study design relates to the work of Madeyski, since we are also measuring the code coverage and the mutation score indicators. Main difference with our study design is the enforcement of programing interface, which will allow us to execute test cases of the individual participant on the code of other participants of the experiment. This, as a result, will provide us with an additional quality attribute of test cases.

III. Experimental Design

The study design of the experiment is described in this section. Specific details of the study (instruction material and code skeleton) can be found at the first author's website[1].

A. Research Question

We defined the following research question as a starting point of the experiment study design:

> *Is there a significant difference between the quality of test cases, produced using test-first and test-last approaches?*

B. Goals of the Experiment

The main goal of the experiment is to compare test case effectiveness when developing software using test driven development (test-first) and traditional (test-last) approaches. Additionally, we measure efficiency of the experiment participants in order to investigate if there is a relation between time spent on solving the problem using specific approach and the quality of the resulting test cases. To confirm a common belief that TDD improves code quality, we are also investigating this attribute. Negative test cases, or test cases constructed from non-specified requirement, were identified as a main problem of our previous TDD experiment [11]. In literature, this phenomenon is known as a positive test bias [12] [13] were testing is done using more positive or specification defined inputs. With this experiment, we are also investigating the ratio of positive and negative test cases for test-first and test-last approaches.

C. Quality Attributes

In our research question, we are using *quality* of test cases as an attribute for comparison of software development approaches. Quality of test cases (or quality of a test suite) is defined as a sum of individual test cases quality. Individual quality can be calculated using three different indicators: (i) code coverage, (ii) mutation score and (iii) total number of failing assertions.

1) Code coverage
   Using EclEmma [14] (Java Code Coverage tool for Eclipse) a coverage score indicator of the created test cases is obtained for each individual solution.
2) Mutation score
   Using Judy [15] (Java mutation tool) a mutation score indicator of the created test cases is obtained for each individual solution.
3) Total number of failing assertions
   Since participants were obliged to implement software solutions using predefined program interface, it is possible to execute test cases of one participant onto the code implementations of other participants. Quality of tests, in the form of a total number of failing assertions, could be defined for an individual experiment participant as:

$$Q_{defects}(S_i) = \sum_{j=1}^{n} DF(TS_i, CS_j)$$

   where, S_i - individual participant of the experiment, DF - defects found, TS_i - test set of participant i, CS_j - code solution of participant j, and n - total number of experiment participants.

D. Experiment Design

As a final laboratory work within the course, subjects were given a task to completely implement and test (to the extent they consider sufficient) a bowling game score calculation algorithm. They were grouped in two groups, the test-first and the control (test-last) group. The Eclipse [16] integrated development environment (IDE) was used to create software solution in the Java programming language and the jUnit [17] testing framework was used for writing executable tests. After completely finalising their implementations, the subjects answered a set of questions using an online survey system.

[1] http://www.mrtc.mdh.se/~acc01/testqualityexperiment/

E. Subjects

The subjects of the experiment were software engineering master students enrolled in the Software Verification and Validation course at Mälardalen University during the autumn semester of 2011. The experiment was part of the laboratory work within the V&V course, and the subjects earned credits for their participation. Students were informed that the final grade for the course will be obtained from the written exam and their performance during the laboratory work will not affect the final grade, although they had to complete the laboratory work.

F. Objects

The experiment used a bowling game score calculator problem for the experiment. The specification for this was based on the Bowling Game Kata (i.e., the problem also used by Kollanus and Isomöttönen to explain TDD [18]). Detailed information about the problem and instructions are provided on first author's webpage[2].

G. Instrumentation

Participants in the test-first group were instructed to use TDD to develop software solutions. Instructions for TDD were given as prescribed by Flohr and Schneider [19]. Participants in the test-last (control) group were instructed to use traditional (test-last) approach for software development.

To avoid problems with subjects' unfamiliarity with the jUnit testing framework and/or Eclipse IDE, subjects were given an Eclipse project code skeleton with one simple test case. Since this was all located in a subversion (SVN) repository, an instruction on how to obtain code from an SVN and import it into the Eclipse was also provided to students.

H. Data Collection Procedure

As part of the instructions, subjects were instructed to upload their source code in a subversion (SVN) repository on a regular basis (after each test case was created). This way we have a complete log of subjects activities with an option to obtain the code from a specific point in time. As an addition to providing the source code, subjects answered a set of survey questions using quiz assignments in the Blackboard learning management system used during the course. Data from the survey was then exported in a comma separated values (.csv) file format.

IV. Execution

Once experiment study design was defined and all lectures within Verification & Validation (V&V) course were completed, the pre-requirements for the experiment execution were in place.

[2]http://www.mrtc.mdh.se/~acc01/testqualityexperiment/

A. Sample

Fourteen students participated during the last laboratory work of the V&V course. Students were informed that their lab work would be used for the experiment, but they were not provided any details on the goal of the experiment itself. Also, we explicitly stated that their performance during the lab will not influence the final grade of the V&V course in any way. The final grade was determined by a written exam.

B. Preparation

Subjects worked individually on the implementation and were randomly assigned to follow either test-first or test-last approach. Laboratory work was not time-boxed and subjects were given an opportunity to work on the implementation until they have enough quality confidence in the submitted solution. Since one way of measuring the quality of test cases was using a total number of failing assertions, a Java code skeleton was created and provided to subjects to enforce usage of same programming interface which would ease the process of executing test cases of subject *X* on the code of subject *Y*.

For each student a corresponding subversion (SVN) repository was created with read/write permissions assigned only to a specific student and to the researchers. To avoid difficulties in setting up an SVN and importing project in the Eclipse, an instruction on usage of an SVN and the Eclipse was provided to subjects.

C. Data Collection Performed

Using an SVN tool, researchers have a complete log of activities during the experiment with the ability to obtain the source code at any given point in time. After each student reported to researchers to have a complete solution created and saved in an SVN, the test cases and the code for each student solution was saved.

V. Analysis

This section provides an analysis of the collected data. For each quality attribute a referencing table with data is listed and described.

Table I lists code coverage score in percentage for each experiment participant. This is calculated by a number of statements that accompanied tests reach for a given participants code. In average, coverage that was achieved by test-first and test-last group is nearly the same and relatively very high. This, as a result, makes it difficult to reason or derive any conclusions of the quality of groups test cases by using this quality.

Table II represents values of mutation score indicator in percentage. By using the Judy [15] tool, mutation score is calculated for each participant using next approach:

1) Compiled source code is taken as an input
2) Compiled test cases (test suite) are taken as well as an input, but with the pre-requirements that they are all passing for the original source code.

Test-First Group		Test-Last Group	
	Coverage in %		**Coverage in %**
TF1	96,21%	TL1	100,00%
TF2	97,76%	TL2	91,22%
TF3	100,00%	TL3	89,42%
TF4	83,28%	TL4	98,83%
TF5	98,78%	TL5	98,98%
TF6	98,88%	TL6	95,93%
TF7	99,45%	TL7	98,86%
Average:	**96,34%**	**Average:**	**96,18%**
Median:	**98,78%**	**Median:**	**98,83%**
Std. Dev.:	**5,89%**	**Std. Dev.:**	**5,89%**

TABLE I
CODE COVERAGE SCORE IN PERCENTAGE

Test-First Group		Test-Last Group	
	Mutation in %		**Mutation in %**
TF1	78,35%	TL1	91,30%
TF2	82,11%	TL2	84,31%
TF3	92,57%	TL3	80,58%
TF4	57,54%	TL4	89,19%
TF5	89,80%	TL5	84,52%
TF6	87,83%	TL6	77,65%
TF7	85,12%	TL7	75,49%
Average:	**81,90%**	**Average:**	**83,29%**
Median:	**85,12%**	**Median:**	**84,31%**
Std. Dev.:	**11,75%**	**Std. Dev.:**	**5,80%**

TABLE II
MUTATION SCORE IN PERCENTAGE

3) Using a set of default mutation operators, N number of variations of original program is generated (they are called mutants)
4) For each mutant $n \in N$ the test suite is executed
5) If any test case within the test suite fails, current mutant n is marked as "killed".
6) Total number of killed mutants is m ($m \leq |N|$).
7) Mutation score is calculated as $m/|N|$

Once again, average mutation quality attribute indicators for both groups were similar, thus making it difficult to identify if the test cases of any group could be considered of a better quality.

Table III list a total number of failing assertions found by test cases of each participant. For example, test cases of participant TF4 (which was using test-first approach during the experiment) contained 38 failing assertions when executed on every code instance created by *all* other participants (test-first and test-last). Interestingly, the sum of total number of failing assertions for both groups is nearly the same.

Table IV also provides a total number of failing assertions, but in this case the focus is on the subjects source code. If we look again at participant TF4, there was a total of 17 assertions that failed on the code of this participant from test cases of *all* other participants. Sum of all defects found in the code is lower for test-first participants.

Table V lists duration time needed for participants to fully implement software solution (including testing of it) for the given problem. On an average, test-first group finalised implementation one hour before test-last group.

Table VI provides ratio of positive vs. negative test cases in

Test-First Group		Test-Last Group	
	# of defects found		**# of defects found**
TF1	13	TL1	30
TF2	144	TL2	86
TF3	17	TL3	34
TF4	38	TL4	72
TF5	44	TL5	17
TF6	14	TL6	20
TF7	14	TL7	31
Sum:	**284**	**Sum:**	**290**
Median:	**17**	**Median:**	**31**
Std. Dev.:	**47,33**	**Std. Dev.:**	**26,68**

TABLE III
DEFECTS FOUND BY TESTS

Test-First Group		Test-Last Group	
	# of defects found		**# of defects found**
TF1	57	TL1	24
TF2	3	TL2	11
TF3	58	TL3	55
TF4	17	TL4	9
TF5	10	TL5	72
TF6	55	TL6	143
TF7	44	TL7	16
Sum:	**240**	**Sum:**	**330**
Median:	**44**	**Median:**	**24**
Std. Dev.:	**24,03**	**Std. Dev.:**	**48,53**

TABLE IV
DEFECTS FOUND IN CODE

Test-First Group		Test-Last Group	
	Duration		**Duration**
TF1	05:50:00	TL1	07:13:00
TF2	02:42:00	TL2	05:25:00
TF3	05:44:00	TL3	03:53:00
TF4	03:57:00	TL4	03:27:00
TF5	03:58:00	TL5	09:29:00
TF6	03:53:00	TL6	02:45:00
TF7	05:37:00	TL7	05:38:00
Average:	**04:31:34**	**Average:**	**05:24:1**
Median:	**03:58:00**	**Median:**	**05:25:00**
Std. Dev.:	**01:12:27**	**Std. Dev.:**	**02:21:02**

TABLE V
DURATION OF THE COMPLETE DEVELOPMENT TIME

Test-First Group		Test-Last Group	
	Positive bias in %		**Positive bias in %**
TF1	22,22%	TL1	33,33%
TF2	43,33%	TL2	33,33%
TF3	8,33%	TL3	40,00%
TF4	35,71%	TL4	20,00%
TF5	35,29%	TL5	46,67%
TF6	9,09%	TL6	27,27%
TF7	9,09%	TL7	17,65%
Average:	**23,30%**	**Average:**	**31,18%**
Median:	**22,22%**	**Median:**	**33,33%**
Std. Dev.:	**14,88%**	**Std. Dev.:**	**10,41%**

TABLE VI
RATIO OF POSITIVE VS. NEGATIVE TESTS

participants test suites (given as a percentage). By a positive test case, we refer to an assertion that is testing a functionality in the program which is *defined* in the specification. Negative test cases represent assertions which are testing *non-specified* functionality.

VI. Interpretation

A. Evaluation of Results and Implications

The pilot experiment performed within this study does not have a sufficient number of participants, and thus makes it difficult to draw any conclusions based on a statistical significance of the collected data. However, for several quality attributes, we can observe similarities to the result of the related study [10]. Mainly, we can relate that difference in test cases between the test-first and the test-last participants almost do not exists, if code coverage and mutation score indicators are used for comparison. Interestingly, additional quality attribute that we introduced (the total number of failing assertions) also could not make any distinction between the test-first and the test-last participants.

Data is pointing out that the code of test-first group is of better quality as compared to that of the test-last group. This is an interesting observation considering that both the groups had same quality of test cases used to test the implementation. However, even though test-first group has less failing assertions than test-last group, both groups still have a relatively high number of errors in the code. Test-first participants had more positive test cases than negative, probably as a result of the "inherent"positive test bias of TDD.

B. Limitations of the Study

Similarly to the previously published experiments on TDD [8], this pilot experiment was also performed in an academic setting. Therefore, validity is threatened with some known academic limitations: (i) using students as subjects, (ii) using small scale objects of investigation, and (iii) having short duration of the experiment.

VII. Conclusions and Future Work

In its nature, test-driven development is a development methodology and not a test design technique. This can be often confused since, when using TDD, developers do create a set of automated test cases. Researchers are mostly investigating benefits of using TDD with respect to the quality of produced code, but only one study (to the best of our knowledge) focuses on investigating the quality of the produced test cases.

In this paper we presented a detailed study plan and a pilot experiment on the investigation of test cases quality in TDD. Since there is no underlying theory behind TDD, it is not possible to formally validate its claimed benefits by any other means except performing empirical investigations. In a long term investigation process, this study should be conducted as a fully controlled experiment, with a higher number of participants in order to validate statistical significance of the presented results. Preferably, this should be done in an industrial context investigating if test cases of TDD developers can detect same faults as when using test-last approach.

Acknowledgments

This work was supported by SWELL (Swedish software Verification & Validation ExceLLence) research school.

References

[1] K. Beck, *Extreme programming explained: embrace change*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[2] A. Causevic, D. Sundmark, and S. Punnekkat, "An industrial survey on contemporary aspects of software testing," in *Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST)*, 2010, pp. 393–401.

[3] B. George and L. Williams, "A structured experiment of test-driven development," *Information and Software Technology*, vol. 46, no. 5, pp. 337 – 342, 2003.

[4] H. Erdogmus, M. Morisio, and M. Torchiano, "On the effectiveness of the test-first approach to programming," *IEEE Transactions on Software Engineering*, vol. 31, pp. 226–237, 2005.

[5] D. S. Janzen and H. Saiedian, "On the influence of test-driven development on software design," *Software Engineering Education and Training, Conference on*, vol. 0, pp. 141–148, 2006.

[6] A. Gupta and P. Jalote, "An experimental evaluation of the effectiveness and efficiency of the test driven development," in *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 285–294.

[7] J. H. Vu, N. Frojd, C. Shenkel-Therolf, and D. S. Janzen, "Evaluating test-driven development in an industry-sponsored capstone project," in *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 229–234.

[8] A. Causevic, D. Sundmark, and S. Punnekkat, "Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review," in *Proceedings of the 4th International Conference on Software Testing, Verification and Validation (ICST)*, 2011.

[9] A. Jedlitschka and D. Pfahl, "Reporting guidelines for controlled experiments in software engineering." in *Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE 2005)*, R. J. et al., Ed. IEEE Computer Society, 2005, pp. 94–104.

[10] L. Madeyski, "The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment," *Inf. Softw. Technol.*, vol. 52, pp. 169–184, February 2010.

[11] A. Causevic, D. Sundmark, and S. Punnekkat, "Impact of Test Design Technique Knowledge on Test Driven Development: A Controlled Experiment," in *Agile Processes in Software Engineering and Extreme Programming - 13th International Conference, XP 2012, Malmö, Sweden, May 20-25, 2012. Proceedings*, ser. Lecture Notes in Business Information Processing. Springer, 2012 (to appear).

[12] B. E. Teasley, L. M. Leventhal, C. R. Mynatt, and D. S. Rohlman, "Why Software Testing Is Sometimes Ineffective: Two Applied Studies of Positive Test Strategy," *Journal of Applied Psychology*, vol. 79, no. 1, pp. 142 – 155, 1994.

[13] L. M. Leventhal, B. Teasley, D. S. Rohlman, and K. Instone, "Positive Test Bias in Software Testing Among Professionals: A Review," in *Selected papers from the Third International Conference on Human-Computer Interaction*. London, UK: Springer-Verlag, 1993, pp. 210–218.

[14] EclEmma - Java Code Coverage for Eclipse, http://www.eclemma.org.

[15] Judy - Java mutation tester, http://www.java.mu.

[16] Eclipse, http://www.eclipse.org.

[17] jUnit Framework, http://www.junit.org.

[18] S. Kollanus and V. Isomöttönen, "Understanding tdd in academic environment: experiences from two experiments," in *Proceedings of the 8th International Conference on Computing Education Research*, ser. Koli '08. New York, NY, USA: ACM, 2008, pp. 25–31.

[19] T. Flohr and T. Schneider, "Lessons learned from an xp experiment with students: Test-first needs more teachings," in *Product-Focused Software Process Improvement*, ser. Lecture Notes in Computer Science, J. Mnch and M. Vierimaa, Eds. Springer Berlin / Heidelberg, 2006, vol. 4034, pp. 305–318.