Effects of Negative Testing on TDD: An Industrial Experiment

Adnan Causevic[1], Rakesh Shukla[2], Sasikumar Punnekkat[1], and Daniel Sundmark[1]

[1] Mälardalen University, Sweden
{adnan.causevic,sasikumar.punnekkat,daniel.sundmark}@mdh.se
[2] Infosys Ltd. India
rakesh_shukla@infosys.com

Abstract. In our recent academic experiments, an existence of positive test bias, that is lack of negative test cases, was identified when a test driven development approach was used. At the same time, when defect detecting ability of individual test cases was calculated, it was noted that the probability of a negative test case to detect a defect was substantially higher than that of a positive test case.
The goal of this study is to investigate the existence of positive test bias in test driven development within an industrial context, and measure defect detecting ability of both positive and negative test cases. An industrial experiment was conducted at Infosys Ltd. India, whose employees voluntarily signed up to participate in the study and were randomly assigned to groups utilizing test driven development, test driven development with negative testing, and test last development. Source code and test cases created by each participant during the study were collected and analysed.
The collected data indicate a statistically significant difference between the number of positive and negative test cases created by industrial participants, confirming the existence of positive test bias. The difference in defect detecting ability of positive and negative test cases is also statistically significant. As a result, similarly to our previous academic study, 29% of all test cases were negative, contributing by revealing as much as 71% of all the defects found by all test cases. With this industrial experiment, we confirmed the existence of a positive test bias in an industrial context, as well as significantly higher defect detecting ability of negative test cases.

Key words: Test-driven Development, Industrial Experiment, Quality of Testing

1 Introduction

Performing efficient and effective software testing often comes with many challenges. Increased complexity of software systems, the need for a specific domain knowledge or the lack of testing experience are just a few obstacles a tester is

faced with in day to day activities. Today, with the presence of Agile methods, the quality of the software product becomes everyone's responsibility, not just the quality assurance or the testing department. A potential problem here is that not every member of the team has the appropriate expertise and sufficient training in quality assurance methods.

Test driven development (TDD) is one example of how developers can focus on the quality of software by writing executable and automated test scripts before writing the actual code. TDD was introduced as part of the eXtreme Programming (XP) methodology [1]. By writing test cases before the code, developers use tests to guide them in the correct implementation of the required functionality. In the literature, TDD is also referred as a test-first approach [2].

TDD was identified, in our industrial survey [3], as a preferred but not often used practice in industry. An interpretation of this finding could be that "respondents would like to use TDD to a significantly higher extent than they actually do". One reason for this preference towards TDD, could be that academic research results often highlight improvements in the code quality when TDD is utilized [4–8]. In our further investigations, a systematic literature review [9] was performed for the purpose of identifying any obstacles for a full scale adoption of TDD in an industrial context. Developer's inability to write efficient and effective automated test cases is considered to be one of the limiting factors. During the autumn semester in 2011, an academic experiment was conducted with master students with the intention of comparing testing efficiency and effectiveness of agile (test-first) and traditional (test-last) developers. This experiment [10] allowed us to investigate various software testing quality metrics in test-driven development by using created test cases, thus investigating the significance of a previously identified limiting factor.

Although we could not find any differences in the quality metrics for testing among the test-first and the test-last group of developers, we did notice that both groups created a much higher number of positive test cases compared to the negative ones. This effect, also known as positive test bias [11, 12], was present for both the test-first and test-last group. Interestingly, when measuring defect detecting ability of test cases, negative test cases detected much more defects compared to the positive ones. But since this was a limited study in an academic context, a larger study in an industrial context was needed to confirm the external validity of our findings and to verify if such an effect exists in the industry and to what extent, when compared to the academic results. Additionally, we want to investigate how test-driven development could utilize creation of negative tests while still "driving" the development. In this paper we present the results of such an industrial experiment.

2 Related Work

Empirical studies, performed for the purpose of investigating potential benefits of TDD focus mainly on the differences in the quality of the produced code. This was one of the finding we noticed in our systematic literature review [9], where we listed 48 empirical studies that had effects of TDD as the focus of the investigation. In most cases, TDD was in the primary focus of the investigation, but in some studies TDD was used together with a different practice, e.g. pair-programming.

However, one study was identified with the focus on quality attributes of *test cases* when a test-first approach was used. Madeyski [13] investigated how usage of TDD can impact branch coverage and mutation score indicators. In his experiment, 22 students were divided in two groups: the test-first and the test-last, with the task of developing a web based conference paper submission system. This experiment shows no statistically significant differences in branch coverage and mutation score indicators, between the test-first and the test-last groups. To the best of our knowledge, after performing our systematic review, we noticed one additional study [14] with the focus on developer's testing ability when following test driven development approach. This was an industrial observational experiment where developers performed programming tasks in their own offices without the control of researchers. Once developers submitted their code and test cases, researchers performed mutation testing to identify complementary test cases to the ones created during TDD process. Those unit tests, created by researchers, were still able to find several software faults in the submitted code.

In our previous academic study [10] we measured the code coverage and the mutation score indicators in a very similar way as it was done by Madeyski in his study. In accordance with the results of Madeyski, we were not able to notice any differences between the experiment groups. For the experiment presented in this paper, we have opted not to measure and analyse those attributes. The reason is that both those indicators are considered as an internal quality attributes of a test suite, while we are more interested in measuring the actual effectiveness of a test cases with respect to the defect detection. Specially, this is useful to distinguish what effect positive and negative test cases have on the overall testing effort. This is why our study design enforces programming interface for all participants, allowing us to execute the test cases of one participant on the code of all other participants.

Having results from our previous studies pointing out that experiment participants have a very small focus on "negative" test cases (existence of a positive test bias), the experiment performed in this study has a built-in mechanism of differentiating whether a particular test case is of a positive or negative type. More detailed explanation of what exactly constitute a negative test case, how we measure defect detecting efficiency and other concepts introduced in this study are presented in the following section.

3 Methodology and Study Design

Before going into the details of the experiment design, its execution and the analysis of the collected data, we would like to present to the reader several concepts which are defining the methodology used this study.

Negative Testing
By the term *negative test case*, we refer to a test case that was created for the purpose of exercising a program in a way that was not explicitly specified in the requirement. On the other hand, a *positive test case* exercises a program behaviour as it is specified in the requirement.

For example, a specification might state: "*... numbers are accepted as an input to the program ...* ", and testing such a program with any numerical input is considered as positive testing. For the same program, if testing is performed by providing a *character* input, that can be called negative testing.

Even in the case of an implicit specification, for example: "*... only numbers are accepted as an input to the program ...* ", testing using *character* inputs can still be considered negative testing, unless it is explicitly stated in the specification how the program should behave upon receiving character inputs.

Quality of Testing
Quality of software has been an active research area for the past few decades and several software measures ranging from simple lines of code to various complexity measures can be found in the literature to evaluate and improve the software quality from process or product perspectives [15]. However there had been no consensus on the universal applicability of any specific software quality metric and their usage had been more context specific and based on the intended objectives.

For our study we are primarily interested in formulating measures that can help in evaluating and improving the quality of testing, a topic which so far has not received much research attention. Obviously quality of testing is strongly related to the ability of test cases in finding defects in the code. However, in the context of new development paradigms like TDD, test cases are created by developers more as a safety net of the implemented functionality. They are capable of detecting wrong changes on the current software implementation, but they are not primarily focussing on finding defects. As a result the final set of test cases that accompany a software solution developed using TDD will show only its correctness but no defects in the same. In order to realistically measure the quality of testing we need to essentially have access to an *ideal test suite* which is capable of finding all the defects. Our approach here will be to approximate such an *ideal test suite* by combining all the test suites developed by several individual developers working on the same problem. Given such a set of multiple implementations and associated test suites, we are then able to cross-compare the ability of test cases to find defects.

Defect Detecting Ability
Defect detecting ability represents a total number of defects a particular test case can find in all the implementations of the same problem created by different developers. This number could be also calculated for all test cases created by a single developer, but more interestingly in the context of this study is to calculate how many defects are detected by negative and positive test cases.

Additionally, considering the differences in the expertise levels of the developers, we would like to give a higher quality value to a test case that is capable of finding defects in an implementation of high quality. Hence the evaluation of the quality of test cases will be much more meaningful if we jointly address it together with the quality of code in which we apply them.

Quality of Code (Q_{code})
Main reason why we need to calculate quality of code attribute is to support calculation of *Quality of Tests* attribute. Quality of the code for every developer (i) is calculated using next formula:

$$Q_{code}(i) = 1 - \frac{N_{FTC}(i)}{N_{TC}}$$

where, N_{TC} is a total number of test cases created by all developers and $N_{FTC}(i)$ represent total number of failing test cases on the code of a developer (i) by executing all test cases from all developers. Once we calculated *Quality of Code* value for each developer, we can now reward test cases who are able to detect defects in the underlying code.

Quality of Tests (Q_{tests})
Quality of test cases for a developer (i) is calculated as a sum of the quality of each test case (j) from a set of test cases (n) of that developer (i):

$$Q_{tests}(i) = \sum_{j=1}^{n} Q_{TC}(i.j)$$

To calculate the quality of an individual test case (j) of a developer (i) we need to know on which developers' code this test case is failing ($m \in M$). Sum of the *Quality of Code* values (Q_{code}) of those developers will define the quality of a particular test case (j):

$$Q_{TC}(i.j) = \sum_{k=1}^{m} Q_{code}(k)$$

Once this calculation is done for every developer, we can have a much better understanding of how much each and every test case contribute to the overall quality of testing. In the context of this study, it is interesting to observe how much negative and positive test cases contribute to the quality of testing.

3.1 Study design

The design of this experiment originates from the academic study elaborated in [16]. However, to accommodate all challenges with performing such an experiment in industrial context (professional developers in an industrial settings), some modifications to the academic study design had to be done.

This experiment was setup with several goals in mind. As the main goal, we wanted to investigate if the effect of a positive test bias could be identified in an industrial setting regardless if the *test last* approach or TDD was used. We wanted to be able to examine the existence of such an effect by calculating the effectiveness of the provided tests. As an additional goal, in case the positive test bias effect existed, we wanted to investigate if it is possible to eliminate such an effect by providing participants with the support for the negative tests. Based on these goals, the following research questions were defined:

RQ1: *Does the effect of positive test bias exists in an industrial context?*
RQ2: *Is the defect detecting ability of negative test cases the same as the positive ones?*
RQ3: *Is the quality of negative test cases the same as that of positive test cases?*
RQ4: *Is there a difference in the quality of produced tests based on the usage of a specific development practice?*

In order to perform statistical testing, with respect to the stated research questions, following null and alternate hypotheses were formulated:

$\mathbf{H}^1{}_0$ There is no difference between the total number of positive and negative test cases created by experiment participants.
$\mathbf{H}^1{}_a$ There is a difference between the total number of positive and negative test cases created by experiment participants.
$\mathbf{H}^2{}_0$ There is no difference between the number of failing assertions detected by positive and negative test cases.
$\mathbf{H}^2{}_a$ There is a difference between the number of failing assertions detected by positive and negative test cases.
$\mathbf{H}^3{}_0$ There is no difference between the quality of positive and negative test cases.
$\mathbf{H}^3{}_a$ There is a difference between the quality of positive and negative test cases.
$\mathbf{H}^4{}_0$ There is no difference in the quality of testing based on the usage of a specific development practice.
$\mathbf{H}^4{}_a$ There is a difference in the quality of testing based on the usage of a specific development practice.

4 Experiment Design

The experiment was performed within Infosys Ltd.[1] India in September 2012, as part of the Infosys InStep[2] internship program. Participants of this experiment are Infosys employees spread around several development centres in India and even some employees participating from the Infosys client's (on-site) locations. The set of participant locations include: Bangalore, Beaverton, Brussels, Chennai, Hyderabad, Mangalore, Melbourne, Mysore, Pune, Trivandrum.

Since the participants of the experiment were not located at one single place, we have opted for a semi-controlled version of the experiment design, compared to the previous fully controlled academic study. This way, participants did not have to be physically present at the same time in the same room to perform their experiments under a supervisor in a controlled environment. Additionally, participants could decide when it was most convenient for them to work on the experiment task, based on their current project related duties, deliverables or deadlines.

At first, the researchers needed to create training materials for each group of participants: (i) Test Last (TL); (ii) TDD; and (iii) TDD with the Support for Negative Testing (TDD+). Upon receiving the list of the experiment participants, the researchers randomly divided them into the three previously mentioned groups. The number of participants in each group was kept as equal as possible. Using the Microsoft SharePoint[3] infrastructure, the training material was distributed by creating a SharePoint Workspace folder for each individual participant. In order for participants to access their SharePoint Workspaces, the researchers sent an invitation email to each participant. Each participant worked individually on an implementation of a defined problem in Java using the Eclipse [17] integrated development environment (IDE). Test cases were written using the jUnit [18] testing framework. The implementation was enforced with the provided programming interface. Upon completing their tasks, the participants of the experiment updated their SharePoint Workspace with their solution to the problem.

The *Bowling Game Score Calculator* problem was used for the experiment. The specification was based on the Bowling Game Kata (i.e., the problem also used by Kollanus and Isomöttönen to explain TDD [19]). From our experience, on average, 3 hours are needed to fully implement the problem and usually around 10 test cases are created during implementation when following TDD approach. Detailed information about the problem and instructions are provided on the first author's webpage[4].

Participants assigned to the TDD group were instructed to use TDD steps to develop software solution. Instructions for TDD were given as prescribed by Flohr and Schneider [20]. Participants assigned to the TDD+ group were

[1] `http://www.infosys.com`
[2] `http://www.infosys.com/instep`
[3] `http://sharepoint.microsoft.com`
[4] `http://www.mrtc.mdh.se/~acc01/infosys-experiment`

instructed to use the same TDD steps with the addition to the very first step. Basically, they were instructed to follow TDD, but also to occasionally write a negative test case based on the input space, domain knowledge, etc. (but not explicitly stated in the requirements). Participants assigned to the *Test Last* (TL) group were instructed to use traditional (test-last) approach for software development and they were considered as a control group for this experiment.

It was expected that some participants may not be familiar with the usage of the jUnit testing framework and/or Eclipse IDE. To avoid such problems, video tutorials were created for each group of participants. Additionally, participants were given an Eclipse project code skeleton which included one simple test case. Specific details of the study, i.e. instruction material, video tutorials and code skeleton, can be found at `http://www.mrtc.mdh.se/~acc01/infosys-experiment`.

The participants were instructed, upon finalizing their software implementation, to save the source code together with the test cases in their individual and predefined SharePoint Workspace. Additionally, participants had to complete a simple questionnaire stating their opinions on the quality of the provided solution.

5 Execution

The first author of this paper stayed at Infosys development center in Bangalore, India, from 3rd to 28th of September 2012. During the first week of internship it was decided that the experiment will be executed from 10th till 21st of September, 2012. First week was used to prepare training video material, instructions and survey questions for the experiment.

A list of over 100 email addresses of Infosys employees was previously obtained by the second author, who directly promoted this experiment among employees of Infosys. Those employees were randomly distributed in three groups: TL, TDD and TDD+.

Participants were informed that their enrolment in this experiment would support the current research activities within Infosys, but the exact details of the experiment, as well as the goal of the experiment, were not shared with them. Additionally, participants were explained that their source code and test cases would be analysed anonymously and this activity will not be used in any way for the internal employee evaluation.

For each participant, a SharePoint Workspace was created with the dedicated training material and instructions placed in it. As an alternative files could be obtained from Internet. In particular, Eclipse IDE should be obtained as well as an experiment instruction document. Video tutorials were also hosted outside of Infosys intranet and the link was provided to participants. In some cases, Microsoft Office Communicator was used to transfer the required files.

Participation in the experiment was not time-boxed and subjects were given an opportunity to work on their implementations until they have enough confidence in the quality of 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 the same programming interface which would ease the process of executing the test cases of a subject X on the code of a subject Y.

Upon finishing their development task, participants uploaded software solution to their dedicated SharePoint Workspace or sent their solutions by email to the first author.

Table 1 present the number of solutions submitted for the experiment analysis by participants of the experiment.

Table 1. Distribution of solutions per groups

Group	Submitted	Removed	Analysed
TL	19	8	11
TDD	21	10	11
TDD+	20	9	11
Total	60	27	33

Once the submitted solutions were individually inspected (code was visually reviewed and tests were executed), several submitted solutions had to be removed before the analysis. Reasons for removal are listed below:

Incomplete solutions
Manually looking at the code it was possible to identify that some provided solutions were not completed. We can only assume they were submitted as such due to external deadlines.

Own failing test cases
Number of solutions had test cases which were failing on their own code. This was usually a sign of an uncompleted solution.

Small number of test cases (≤ 3)
In case a submitted solution did not have a minimum of 3 test cases (average was 13,8), such would be removed from the analysis.

Wrong test cases
It is very important not to have a false positives in the test cases. In case a test case is expecting a wrong result, the same was removed from the test suite, but if most of the test cases are wrong, then the complete solution was removed from analysis.

Different programming interface
Some solutions used a different programming interface which prevented executing other participants test cases on its code, or executing its test cases on other participants code.

6 Analysis

This section provides the analysis of the collected data. The analysis was performed using the R software environment for statistical computing [21]. Aggregated data and the analysis script for R are provided on the first author's webpage[5].

Positive Test Bias
One of the first thing we wanted to investigate with this experiment is the existence of a positive test bias within our participants as defined in the research question **RQ1**. We used the **Wilcoxon** signed rank test for paired nonparametric data in order to test the $\mathbf{H}^1{}_0$ null hypothesis with $\alpha = 0.05$. With a **p-value of 0.00000731** we can reject the null hypothesis and confirm that a difference between the created number of positive and negative test cases is significantly different for our industrial participants.

Defect Detecting Ability
As previously discussed, it is important to compare defect detecting ability of both positive and negative test cases. By doing that we can explore further how lack of negative tests could potentially affect the overall testing effectiveness as defined in research question **RQ2**. Again, the **Wilcoxon** signed rank test for paired nonparametric data was used in order to test the $\mathbf{H}^2{}_0$ null hypotheses with $\alpha = 0.05$. We can reject the stated hypothesis since the **p-value is 0.00000302**, confirming that there is a significant difference in the efficiency of positive and negative test cases.

Quality of Test Cases
By calculating the quality of tests, as defined in section 3, we are able to compare quality of negative and positive test cases and investigate if there are any differences between them. This additional analysis is important because even if we are detecting more defects with one type of test cases (negative in this case), we need to make sure that those defects are not detected, for example, only in the code of a lower quality. With this analysis we are addressing research question **RQ3**.**Wilcoxon** signed rank test for paired nonparametric data was once again used in order to test the $\mathbf{H}^3{}_0$ null hypotheses with $\alpha = 0.05$. We can reject the stated hypothesis as well, since there is a significant difference (**p-value is 0.00000277**) in the quality of positive and negative test cases.

Quality of Testing
As an additional goal of this study, we wanted to cross-compare the quality of tests created by our participants, regardless of the development approach they were using, thus addressing the research question **RQ4**. The **Mann-Whitney** nonparametric test was used in order to test the $\mathbf{H}^4{}_0$ null hypotheses with $\alpha = 0.05$. We can not reject stated hypothesis (**p-value is 0.4102955**), leading to the conclusion that there is no statistically significant difference in the quality of tests produced by participants using TDD and TL.

[5] `http://www.mrtc.mdh.se/~acc01/infosys-experiment`

7 Interpretation

In this section we discuss the results of our experiment, implications they can have on further research, and potential threats to their validity.

7.1 Evaluation of Results and Implications

Our previous academic experiment [16] was performed with a limited number of participants and although we could see some trends, it was difficult to evaluate statistical significance of the collected data. However, the industrial experiment presented in this study enabled us to perform hypothesis testing using statistical methods on the data we collected.

With the industrial experiment data we can confirm the following hypothesis:

$\mathbf{H}^1{}_0$ There is a difference between the total number of positive and negative test cases created by experiment participants.

$\mathbf{H}^2{}_0$ There is a difference between the number of failing assertions detected by positive and negative test cases.

$\mathbf{H}^3{}_0$ There is a difference between the quality of positive and negative test cases.

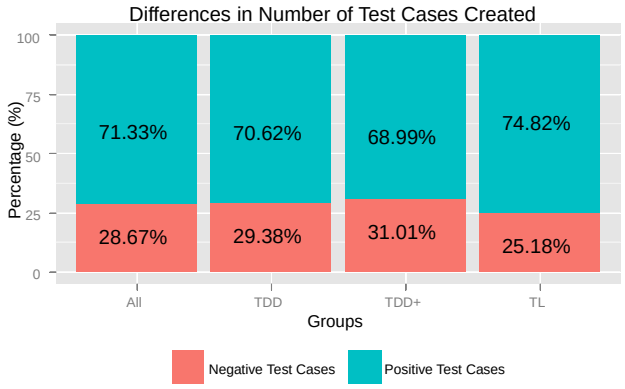

Fig. 1. Number of Test Cases

When looking at the actual differences in the number of created positive and negative test cases, as shown in Figure 1, we can notice that a very similar 70%-30% ratio exists for all groups individually as well as for all participants test cases combined together. Although this may sound like a problem itself, there is no actually (to the best of our knowledge) scientifically proven optimal ratio of positive and negative test cases. The ratio of 70%-30% might work just fine. This is why we emphasize on measuring the defect detection ability of test cases.

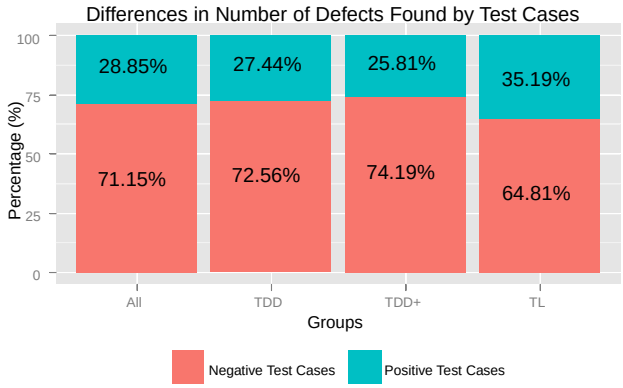

Fig. 2. Defects found by Test Cases

Figure 2 represents the ratio of defects found by positive and negative test cases. There are two interesting observation with this figure: (i) the ratio seems to be the opposite (30%-70%) and (ii) the data presented in this graph is not normalised. When we combine those two observations, we can see that with less than 30% of test cases in our test suite (negative test cases), we are discovering as much as 70% of all the defects detected by a complete test suite.

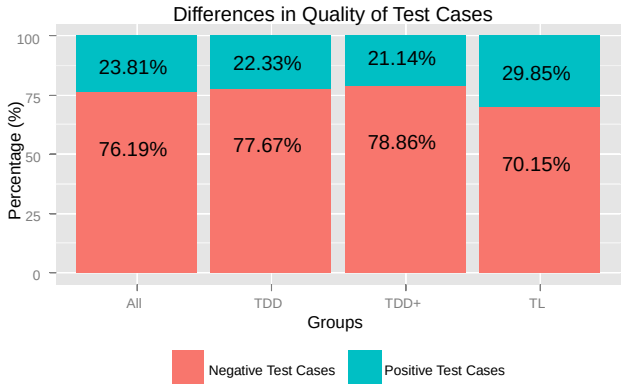

Fig. 3. Quality of Test Cases

As a final step in confirming significance of the effectiveness of negative test cases, Figure 3 presents the ratio to what extent positive and negative test cases are contributing to the overall *Quality of Tests* score for individual groups and for all participants together. As we defined in section 3 of this paper, *Quality of*

Tests measurement was needed since every defect that was detected by a test case was considered of equal importance, which may not represent a realistic situation. By rewarding test cases who are failing on the code of high quality, we could differentiate between the *raw* number of defects detected and the actual quality they bring into the overall testing effort.

The ratio of quality between positive and negative test cases goes beyond of 30%-70%, confirming that negative test cases are detecting more number of defects, but are also detecting defects on the code of a higher quality, resulting in finding defects that are not commonly discovered by positive test cases.

7.2 Threats to Validity - Reservations

The major advantage of this experiment, compared to our previous academic efforts, is the usage of a large number of industrial developers instead of a small number of master students as the experiment subjects. However, setting up an experimental study in an industrial environment brings many challenges for researchers. Inability to have a full control of the experiment represent the major threat to the validity of this study, which was due to the nature of distributed work at our industrial partner. Additional threat to the validity of this study is the usage of a small scale object of investigation. In our experience from previous academic experiments, on average around 3 hours is needed to fully complete the experiment task. Considering project related duties, deadlines and other responsibilities of our industrial participants, we decided that this task should be convenient for this experiment as well. Furthermore, because of several day to day activities of our industrial participants, experiment was not executed in one day but rather kept open for two weeks which could represent another potential threat to the validity of this study. Another possible validity threat could be the lack of a domain knowledge of the object for most of the experiment subjects.

Internal validity of the study was addressed by using statistical tests to perform hypothesis testing, as well as by providing the data as part of this publication, in an aggregated form. Additionally, by providing sufficient information about the experiment design as well as training and instruction materials, we are addressing reliability threats related to the replication of this study.

8 Conclusions and Future Work

Test driven development, by definition, is a development methodology and not a test design technique. Test cases are considered only as an artefact of the development activity and as such, in their nature, tests have to “drive” the development in the positive sense. This is one of the reason we focused our research activities on the analysis of test cases created using TDD approach and their efficiency in terms of defect detection. By identifying specific testing knowledge, complementary to the testing skills of a TDD developer, we would enable developers to achieve higher quality of software products, eventually leading to a higher adoption of TDD in industry.

Based on our academic results from previous experiments as well as results from this industrial study, it is evident that positive test bias (i.e. lack of negative test cases) is present when test driven development approach is being followed. On average, in our studies, around 70% of test cases where positive while 30% where negative. However, this effect was not only constrained to TDD since test last or traditional developers experienced the same problem as well.

When measuring defect detecting effectiveness and quality of test cases, an opposite ratio was present. Effectiveness and quality of negative test cases were above 70% while positive test cases contributed only by 30%. These results made evident what importance negative test cases have as part of a test suite. The challenge is how to intuitively create them without disturbing the "driving" of the development when following TDD. This problem was approached with TDD+ group of participants in this experiment (test driven development group with the support for negative testing). Essentially, we instructed this group to optionally write a negative test case when they consider it convenient. But, based on our results, this did not create any differences. Our reasoning for this could be again the effect of positive thinking that TDD requires, as well as a general problem of when to consider convenient to write a negative test case. Currently, we are investigating the possibility of extending test-driven development with particular test design technique, to facilitate consideration of unspecified requirements during the development to a higher extent and thus minimise the impact of a potentially inherent effect of a positive test bias in TDD.

Acknowledgments

This work was supported by the Infosys InStep[6] internship program and the SYNOPSIS project at Mälardalen University.

References

1. Beck, K.: Extreme programming explained: embrace change. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2000)
2. Koskela, L.: Test driven: practical tdd and acceptance tdd for java developers. Manning Publications Co., Greenwich, CT, USA (2007)
3. Causevic, A., Sundmark, D., Punnekkat, S.: An industrial survey on contemporary aspects of software testing. In: Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST). (2010) 393–401
4. George, B., Williams, L.: A structured experiment of test-driven development. Information and Software Technology **46**(5) (2003) 337 – 342
5. Erdogmus, H., Morisio, M., Torchiano, M.: On the effectiveness of the test-first approach to programming. IEEE Transactions on Software Engineering **31** (2005) 226–237

[6] http://www.infosys.com/instep

6. Janzen, D.S., Saiedian, H.: On the influence of test-driven development on software design. Software Engineering Education and Training, Conference on **0** (2006) 141–148
7. Gupta, A., Jalote, P.: 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. ESEM '07, Washington, DC, USA, IEEE Computer Society (2007) 285–294
8. Vu, J.H., Frojd, N., Shenkel-Therolf, C., Janzen, D.S.: 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) 229–234
9. Causevic, A., Sundmark, D., Punnekkat, S.: Factors limiting industrial adoption of test driven development: A systematic review. In: Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on. (march 2011) 337 –346
10. Causevic, A., Punnekkat, S., Sundmark, D.: Quality of testing in test driven development. In: Quality of Information and Communications Technology (QUATIC), 2012 Eight International Conference on the. (September 2012)
11. Teasley, B.E., Leventhal, L.M., Mynatt, C.R., Rohlman, D.S.: Why Software Testing Is Sometimes Ineffective: Two Applied Studies of Positive Test Strategy. Journal of Applied Psychology **79**(1) (1994) 142 – 155
12. Leventhal, L.M., Teasley, B., Rohlman, D.S., Instone, K.: 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) 210–218
13. Madeyski, L.: The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment. Inf. Softw. Technol. **52** (February 2010) 169–184
14. Shelton, W., Li, N., Ammann, P., Offutt, J.: Adding criteria-based tests to test driven development. In: Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation. ICST '12, Washington, DC, USA, IEEE Computer Society (2012) 878–886
15. Fenton, N.E., Pfleeger, S.L.: Software Metrics: A Rigorous and Practical Approach. 2nd edn. PWS Publishing Co., Boston, MA, USA (1998)
16. Causevic, A., Sundmark, D., Punnekkat, S.: Test case quality in test driven development: A study design and a pilot experiment. In: Evaluation Assessment in Software Engineering (EASE 2012), 16th International Conference on. (may 2012) 223 –227
17. Eclipse. http://www.eclipse.org
18. jUnit Framework. http://www.junit.org
19. Kollanus, S., Isomöttönen, V.: Understanding tdd in academic environment: experiences from two experiments. In: Proceedings of the 8th International Conference on Computing Education Research. Koli '08, New York, NY, USA, ACM (2008) 25–31
20. T. Flohr and T. Schneider: Lessons learned from an xp experiment with students: Test-first needs more teachings. In Mnch, J., Vierimaa, M., eds.: Product-Focused Software Process Improvement. Volume 4034 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2006) 305–318
21. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. (2012) ISBN 3-900051-07-0.