Students' and Professionals' Perceptions of Test-driven Development: A Focus Group study

ABSTRACT

We have conducted a qualitative investigation on test-driven development (TDD) with focus groups in order to develop insights on the opinions of developers using TDD regarding the unintuitive process involved, its claimed effects, as well as the context factors that can facilitate (or hinder) its application. In particular, we conducted two focus group sessions: one with professional developers and another with Master students in Computer Science at the University of Basilicata. We used thematic analysis template (TAT) method for identifying the patterns, themes, and interpretations in the gathered data. The application of this qualitative method allowed us to obtain a number of results that can provide directions for future research. Our results can be summarized as follows: *(i)* applying TDD without knowing advanced unit testing techniques can be difficult; *(ii)* refactoring (one of the phases of TDD) is not done as often as the process requires; *(iii)* there is a need for live feedback in order for the developers to understand if the TDD process is being applied correctly; and *(iv)* the usefulness of TDD hinges on the task and domain to which it is applied to.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

General Terms

Experimentation, Human Factors

Keywords

Focus group, qualitative investigation, test driven development

1. INTRODUCTION

Test-driven development (TDD) is an iterative software development technique where unit-tests are defined before production code. In particular, this technique encourages code development by repeating short cycles consisting of: *(i)* writing a unit test for an unimplemented functionality or behavior; *(ii)* supplying minimal amount of production code to make unit tests pass; *(iii)* applying refactoring where and when necessary; and *(iv)* checking that all tests are still passing after refactoring [2]. The effects of these steps can be summarized as follows: a shift in mindset from test-last approach to test-first approach; developing code only to pass tests; a focus on design quality through refactoring operations; and a growing set of regression test cases as a safety net. It is claimed that TDD leads to better code quality due to its focus on testing, and improves developers' confidence in their source code [1]. A number of quantitative empirical investigations have been conducted on TDD (e.g., [5, 21]). Results are somehow contrasting and inconclusive [22]. Even more surprisingly, TDD has been marginally investigated from a qualitative point of view and from the perspective of the developer [13, 23]. Unlike quantitative investigations, qualitative ones allow gaining an understanding of reasons and motivations behind a given phenomenon [26].

In this study, we want to understand what are opinions of developers using TDD regarding the process itself, its claimed effects, as well as the context factors that can facilitate (or hinder) its application. In this respect, focus groups have the advantage, over other qualitative approaches, of producing interactions by focusing on the role of group rather than on individuals. We conducted two focus group sessions with five professional software developers, and 13 Master students in Computer Science. Professional developers attended a professionalization program in which TDD was presented. The students recently took a course about unit testing and TDD. The first focus group session was aimed at gathering the participants impression regarding their learning experience, whereas the second session aimed at exploring issues related with TDD. We used thematic analysis template (TAT) to analyze the recordings of focus group sessions.

The reminder of this paper is organized as follows: background and related work on the measured and perceived effectiveness of TDD is discussed in Section 2. We present our resarch methodology, e.g. planning and design of our focus groups in Section 3. Results are provided and discussed in Section 4. Section 5 concludes the paper.

2. BACKGROUND AND RELATED WORK

The effectiveness of TDD has been assessed through several quantitative studies, and their results aggregated using systematic reviews and meta-analyses. Contradictory

SAC'16 April 4-8, 2016, Pisa, Italy.

http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

results regarding the effects of TDD on both software products (e.g., defects), and the software developers (e.g., productivity) are reported [25, 20, 17]. Interestingly, one of the secondary studies [4] suggested that the *insufficient adherence to the TDD protocol* and *insufficient testing skills* are among the factors hampering the industrial adoption of TDD.

There is a smaller number of qualitative studies investigating the perception of developers regarding the practise. Mueller and Tichy [16] presented the results of using several XP methodologies, including TDD, within a university course. They reported that TDD was one of the most difficult techniques to adopt as writing test cases before coding was at times considered impractical. Nevertheless, students saw the benefits of TDD and ranked it as the best among the practices used as it improved their confidence. Similarly, Gupta and Jalote [7] report that students seemed to be more confident that the testing effort applied by using TDD would bring better results than in a traditional test-after-code setting. They also identified the need of some upfront design. On the other hand, Pancur et al. [19] reports that students perceived TDD more difficult to adopt in comparison to professionals. Students perceived TDD as a practice that hinders their productivity, efficiency and quality. According to professional software developers, TDD helps in devising a better design, and preventing bugs; however, it does not replace a QA engineer [22]. Moreover, TDD improves confidence by minimizing the fear of breaking existing working parts of the code once a new feature is implemented [6].

While quantitative studies provide objective frameworks for assessing the effectiveness of TDD, their findings were mostly inconclusive. On the other hand, qualitative studies enable more deeper understanding. In this respect, existing qualitative studies relied upon non-interactive research methods such as questionnaires. Our study differs from existing work above with the qualitative research methodology employed, i.e. focus groups, in order to develop a better understanding of the underlying phenomena. Our study enables relatively deeper insights, since our results are based on not only individual perspectives, but also the collective understanding reached through interaction. Further, we included both students and professionals in our study, as previous work highlighted some differences among them.

3. THE FOCUS GROUP

Increased attention in empirical methods has also interested software engineering. A broader range of empirical methods are available in software engineering arsenal so that appropriate methods can be selected and used for each research problem [9].

The focus group method is quick to use and cost-effective to obtain qualitative insights and feedback [10]. It can be defined as a research technique that collects data through group interaction on a topic determined by the researcher [15]. Thus, focus groups are carefully planned discussions that the researcher designs to obtain personal perceptions of individuals (or participants, from here on) arranged in groups on a defined area of interest. Focus groups allow the collection of qualitative information and have the benefits of producing upfront and sometimes insightful information. In addition, this kind of research strategy is fairly inexpensive and fast to use even if it shares weaknesses with other kinds of qualitative methods. For example, results may be biased by group dynamics and sample size (a typical group size should range in between 3 and 12). There are several textbooks and detailed guidelines available on how to plan and run focus groups (e.g., [18]). We present the main steps of our research according to the template suggested by Kontio et al. [10]. This template has been suggested for research in software engineering.

3.1 Defining research problem

Rather than providing quantifiable responses to a specific research question, a focus group study provides a flow of input and interaction among participants related to a given topic of interest [12]. The initial objective of our study is to provide insights regarding the adoption and use of TDD. What are the reasons to use TDD? In what contexts? What does a group think developers can achieve by using TDD? What are possible effects when adopting TDD? In Sections 3.3 and 3.4, we framed those discussion points when planning the second focus groups, for both students and professionals.

3.2 Selecting participants

We conducted two focus groups, one involving five professionals and another involving 13 Master students. As for professionals, they attended a training course for professional adjournment on agile software development. The greater part of this course was devoted to introduce TDD and to apply it on actual cases. The course lasted about eight weeks (with four hours of frontal lessons per week) and included homework and workshops working individually and in pairs. Each homework had to be completed in one week.

Industrial experience of professionals ranged between one and ten years. One of the professionals (i.e., the most experienced developer) had a Master degree in Computer Science, while the others had a Bachelor degree in Computer Science. All the professionals had knowledge of testing approaches and techniques (e.g., integration testing, system testing) before our focus group took place.

Participants in the second focus group were students enrolled into an Informative System (IS) course of the Master program in Computer Science at the University of Basilicata. This course had elements regarding software testing, software development process, software maintenance, agile development techniques with a focus on TDD, regression testing, and refactoring. Homework and classwork were also conducted to let students experiment and use TDD, regression testing, and selected testing framework, namely JUnit.[1] The programming language of reference used throughout the class and for the homework was Java. Students had all passed exams related to the following courses: Procedural Programming, Software Engineering I, Object-Oriented Programming I and II, and Databases. Their prior knowledge can be considered rather homogeneous. The same lecturer held both training and IS courses. Both professionals and students were familiar with TLD. That is, a more traditional development technique, in which unit tests are written after a feature (or a set of related features) are considered completed by developers.

[1] http://junit.org

[2] In this context defined as the code-based properties for creating and maintaining the developed solution.

Table 1: Initial template for TAT analysis

ID	Theme	Discussion
1.1	TDD learning experience	Reveals what were the positive and negative points encountered when learning TDD
1.2	Improvements	Reveals what can be done better to improve the negative points in the learning experience
1.3	Classwork and homework	Reveals the challenges encountered while tackling the assignments used to practice TDD
1.4	Appreciations	Reveals appreciations specifically for other peers or lecturers
2.1	TDD in practice	Reveals challenges regarding the application of the TDD process
2.2	Task specification	Reveals the influence of task and context on the application of TDD
2.3	Effects of TDD	Reveals the perceived effects of TDD on several dimensions

Table 2: Final themes and sub-themes identified after TAT analysis

ID	Theme	Sub-theme
2.1	TDD in practice *(focus on process)*	a) TDD internal process characteristics *(testing, refactoring)* b) External process characteristics *(pair-programming)*
2.2	Specification of tasks *(levels of details)*	c) Differences with TLD d) Nature of the task *(greenfield, legacy)*
2.3	Effects of TDD *(internal quality,[2] external quality,[3] productivity)*	e) Differences with TLD f) Existing experience *(novice, professionals)*

3.3 Planning and conducting sessions

We held two sessions for each focus group (i.e., four sessions in total). The first session of each focus group lasted for 30 minutes. The second session of each focus group lasted for about one hour. Both sessions were conducted on the same day. The topics of the sessions were the same for professionals and students. The first session was mostly a pilot, primarily intended to practice our focus group process. Topics discussed in the first session concerned course and classwork, as well as homework and workshops. During the second session, discussion focused in TDD-specific themes.

We started with an overview of the study objectives and a short introduction of the ground rules for discussions during the sessions. We ensured that the participant's opinions represented actual situations by guaranteeing confidentiality and anonymity of discussions. The sessions were conducted in the Italian language, and audio-recorded so that transcripts could be prepared in order to document the points that were raised. The lecturer of training and IS courses was the facilitator for all the two sessions of each focus group. The role of the facilitator in the sessions was to make sure that important topics were touched, limit discussion's sidetracks, and encourage the participants to express their opinions. The facilitator did not take an otherwise active part in the discussion. In addition to the facilitator, there was also another researcher responsible for the collection of notes from the discussions.

Discussions were semi-structured, with pre-defined themes, but not specific questions. A set of themes were proposed by each researcher and successively compared and internally discussed (Section 3.4). This did not limit the dialog and allowed us to touch all the aspects that participants found more relevant.

3.4 Analysis

Audio recordings and notes taken by the researchers were transcribed and anonymized. They represent the data on which we base our analysis. We used thematic analysis templates (TAT) [8] to analyze the data. Thematic analysis is a qualitative method used to identify patterns, themes, and interpretations in the data [3, 14]. The main difference between TAT and a conventional thematic analysis is the use of template documents. A template is a set of initial themes that are studied. Templates are developed by researchers based on their experience and knowledge of the phenomena under investigation. However, templates are not fixed as their content can evolve as analysis progresses. We utilized TAT because of its flexibility and swiftness [8]. In our case, templates were created on the basis of our previous experience in teaching and researching TDD. The TAT was independently carried out by all researchers manually (i.e., without using specialized software). No disagreement regarding the chosen themes was found. In Table 1, we show the resulting template. The first focus group covered themes 1.1 to 1.4. In particular, theme 1.4 was introduced to deal with social acceptability issues [10]. The second focus group covered themes 2.1 to 2.3. For the second focus groups, the themes emerging from data were consolidated in the final set presented in Table 2. Each of the three main themes focused on a particular facet, reported in parentheses. Each theme was articulated into sub-themes. The focus of the sub-themes are also reported in parentheses.

[3]In this context defined as the thoroughness of the developed solution.

4. RESULTS

The results from the previously identified themes are presented below grouped by session.

4.1 First session

A broad discussion on themes 1.1 to 1.4 is not possible due to both space restrictions and minor relevance with respect to the main focus of this paper.

Theme 1.1) TDD was interesting to learn.

Although professionals and students usually use TLD in their work or in university courses, they found TDD interesting as a different software development technique. They did not exclude the potential use of TDD in the future. This is a positive point related to their learning experience. On

the other hand, participants considered classwork delivery time (3 hours) too short, and felt under pressure for delivery on time. In addition, participants found some classwork/homework requirements unclear.

Theme 1.2) Simple exercises should be used to internalize TDD.

The attendees suggested to choose less complex classwork/ homework with well defined requirements description.

Theme 1.3) Tasks and their specification sometimes made it difficult to accomplish assignments.

Professionals and students encountered different challenges while they tackled the assignments used to practice TDD. Students had no experience in dealing with existing software. This represented a challenge for them because some of the assigned homework involved working with existing software. The English language used to specify the requirements of all assignments was another challenge because both professionals and students usually implement requirements specified in Italian language.

Theme 1.4) Appreciation for the lecturer.

Participants expressed appreciations for the lecturer because they were available to explain both requirements and code.

4.2 Second session

We report the results according to the themes shown in Table 2. For each of the main themes, we report main findings and results emerging from sub-themes. Interestingly, for both themes 2.2 and 2.3 the discussion was concerned with the differences between TDD and TLD.

4.2.1 TDD in practice

Theme 2.1) Process-related characteristics restrain the application of TDD.

a) TDD internal process characteristics
Although the three steps of the TDD process were easily understood, participants agreed that their application presents several hurdles. The first step—writing a test for a non-existing functionality—requires, except for trivial cases, a good knowledge of unit-testing patterns and unit-testing framework. One example is writing a failing unit test for a new feature which depends on another entity, e.g.; another class. The lack of knowledge of test doubles [24] (sometimes referred as impostor pattern) and mocking frameworks can make the application of TDD inherently difficulty. On the other hand, a more traditional approach (e.g., TLD) is not restricting in such regards as it allows developers to implement required dependencies that can be later tested together. Although participants were accustomed to use refactoring techniques, they candidly acknowledged that refactoring is more often neglected. The reasons are: *(i)* the difficulty of identifying refactoring opportunities and *(ii)* the less desirability of refactoring when compared to the more fulfilling task of re-starting the TDD cycle by implementing a test case for a new feature. We can postulate that a better IDE support can be beneficial in both circumstances.

The participants, being freshly introduced to TDD, acknowledged that they could not judge whether TDD was being applied correctly. One of the participants declared that *"it is easy to fool yourself with TDD."* The perceived lack of confidence was suggested to be the reason to fall back to a test-last approach. Tools[4,5] that give live feedback about developer's conformance to TDD exists, but their effectiveness is not sufficiently studied [4]. We can postulate that the use of these tools can be beneficial for TDD novice developers. We advise this point as a possible future direction for TDD research work.
b) External process characteristics
Participants deemed that TDD is regarded as an activity that should be embraced by the whole development team. Participants also practiced TDD in pairs, and recognized it as a good fit for pair-programming.

4.2.2 Specification of tasks

Theme 2.2) The task is critical.

c) Differences with TLD
During training course, professionals deliberately practiced TDD on several tasks of different complexity and nature. The same held for students. Some tasks focused on the implementation of algorithms, others on architectural problems (e.g., focusing on the interaction between components). Some tasks were greenfield (i.e., the task is tackled from scratch), while others were brownfield (i.e., some of the components were already in place and the task consisted in modifying or adding functionalities).

TDD is better suited for smaller tasks, whereas its application to a larger task was found undesirable. On the other hand, in presence of a really simple task or a task for which the developer has good knowledge of the domain, applying TDD or TLD did not seem to matter as personal experience trump the specific technique.

TDD was recognized useful within an unknown domain as it allows the development of more *explorative* solutions; whereas TLD is to be preferred when a comprehensive plan of action is available. Finally, TDD is to be preferred when task requirements are likely to change.
d) Task nature
TDD could help understanding legacy code, but only when a test suite is already in place. Otherwise, TLD is preferred.

4.2.3 Effects of TDD

Theme 2.3) There are tradeoffs between internal and external quality, and productivity.

e) Differences with TLD
The discussion of the effects of TDD also concerned how they vary in comparison with TLD. Participants discussed

[4]Besouro - `http://github.com/brunopedroso/besouro`
[5]Pulse - `https://github.com/sbastn/pulse`

peculiarities of TDD and TLD, rather than trying to articulate which one is better than the other. From the discussion regarding this topic, different outlooks emerged between novice and professional participants. Hence, results regarding sub-theme *f)* are based on the existing experience of participants.

f) Existing experience

Novice developers thought TDD improved their productivity because bugs are promptly found. Alongside, the emphasis of TDD on writing tests, and the high rate at which they are run helps to find bugs not only in production code but also in test code. Secondarily, participants perceived positive effects on external quality. TDD manifested its biggest drawback is in terms of internal quality. The process encourages developers to write *quick-and-dirty* production code to make the tests pass, provided that refactoring is then applied. However, participants acknowledged that refactoring is often ignored. This is considered to be the reason for detrimental effects of TDD on internal quality. On the other hand, TLD is considered to benefit internal quality. The professional participants stressed how TDD is time-taking and detrimental for productivity. Their explanation is that the small increment developed with TDD—without a general idea—made them reconsider their previous implementation decisions and forced them to change parts of the existing code. The result of a meta-analysis [20], focusing on the difference between the application of TDD in industry and academia, supports the claimed drop in productivity. Our rationale is that experienced developers tend to strictly follow the process, whereas novices are more likely to interpolate TDD with TLD.

4.3 Threats to Validity

Focus groups could be prone to problems associated with qualitative data. As the developers of methods/approaches may also act as the researcher responsible for focus group sessions, researcher biases could be present either during the planning, during the sessions themselves, or during the analysis [11]. This kind of bias is not present in our study because we developed neither TDD not TLD. To further deal with possible threats to the validity of our results, we used disciplined, objective, and rigorous instrumentation, and data analysis methods [9]. In addition, all our results are based on traceable data. Other possible threats to the validity are related to focus group weakness [10]: *group dynamics*,[6] *social acceptability*,[7] *hidden agendas*,[8] and *limited comprehension*.[9] As for group dynamics, we used semi-structured discussion techniques and the facilitator balanced discussions and activated less active participants. Social acceptability weakness was mitigated by laying out appropriate ground rules in the beginning. The moderator took his role in driving the discussion in order to avoid as much as possible social acceptability issues. Hidden agendas did not affect our study results because business relationships among participants in each session were not present. In addition, it was clearly communicated to participants that results will be presented in anonymous form. We also emphasized that study results could be important for both academy and industry. As for limited comprehension, we selected participants of equal expertise in each session, namely professionals and students. It is worth mentioning that *secrecy* does not affect the validity of obtained results because relevant information concerned with proprietary or business reasons was not discussed in our focus group sessions.

[6]As a focus group discussion takes place without predefined format, it is possible that group dynamics or communication styles influence the level of activity.

[7]It can influence points made during discussion. For example, it is possible that a participant volunteers incorrect information and disagreement may take place accordingly.

[8]Examples of possible biases are: business relationships between participants, motivation to appear in favorable light or not because of result publication, and internal politics of participants' companies.

[9]Time for discussions is limited and communication happens most often only only verbally during the discussion. Complex issues or points could not be understood by all participants.

5. CONCLUSION AND FUTURE WORK

In this paper, we reported the results of two focus groups in which students and professional software developers discussed their experience carrying out programming tasks of different nature using test-driven development (TDD). We held two sessions for each focus group (i.e., four sessions in total). The first session dealt with high-level topics regarding the courses where participants studied and experimented TDD. The most important result is that TDD was interesting to learn. Both students and professionals agreed on this point. The second session touched more practical aspects related to TDD. We used thematic analysis templates (TAT) when discussing: the challenges that TDD process poses, the context—with a particular focus on the programming task—that can hamper or support the use of TDD, and the different effects TDD has on common aspects, like software quality and developers' productivity. Regarding TDD process, we found that:

- Applying TDD without knowing advanced unit testing techniques (e.g., mocking), can be difficult.
- Refactoring is not done as often as the process requires.
- There is a need for live feedback to ensure that TDD is being applied correctly.
- TDD works better if used in conjunction with pair-programming.

We believe that our results can provide further research opportunities. For example, when studying the efficacy of TDD, the ability of the study's participants to apply advanced testing and refactoring techniques should be taken into account. When studying TDD, we recommend that IDEs to support the process informing the developer when TDD is not being applied correctly.

Regarding the kind of tasks to which TDD is applied, we found that:

- TDD suits small task better.
- TDD is preferred for tasks which domain is not well know, i.e. for exploration.
- TDD is applicable to legacy code already covered by unit tests.

In this regard, one of our future endeavors is to test the hypothesis that TDD works better with high-granular requirements rather than coarse requirements through a controlled experiment. The use of TDD with legacy code also represents a challenge for future research.

Regarding the effects of TDD when compared to test-last development (TLD), we found that:

- Novices believed that TDD improves productivity at the expenses of internal quality.

- TLD yields a better internal quality over TDD.
- Professional deemed that TDD decreases productivity.

We recommend future work to investigate how the application of TDD by developers of different experience impacts software attributes such as internal and external quality, as well as productivity. Finally, we recommend the use of focus groups as a tool to efficiently generate hypotheses, that can be later tested using quantitative methodological approaches.

6. REFERENCES

[1] D. Astels. *Test Driven Development: A Practical Guide*. Prentice Hall Professional, 2003.

[2] Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[3] V. Braun and V. Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[4] A. Causevic, D. Sundmark, and S. Punnekkat. Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pages 337–346. IEEE, 2011.

[5] D. Fucci and B. Turhan. On the role of tests in test-driven development: a differentiated and partial replication. *Empirical Software Engineering*, 19(2):277–302, 2014.

[6] A. Geras, M. Smith, and J. Miller. A prototype empirical evaluation of test driven development. In *Software Metrics, 2004. Proceedings. 10th International Symposium on*, pages 405–416, 2004.

[7] A. Gupta and P. Jalote. An experimental evaluation of the effectiveness and efficiency of the test driven development. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 285–294, 2007.

[8] N. King, C. Cassell, and G. Symon. Using templates in the thematic analysis of texts. *Essential guide to qualitative methods in organizational research*, pages 256–270, 2004.

[9] J. Kontio, J. Bragge, and L. Lehtola. The Focus Group Method as an Empirical Tool in Software Engineering. In *Guide to Advanced Empirical Software Engineering*, chapter 4, pages 93–116. Springer London, London, 2008.

[10] J. Kontio, L. Lehtola, and J. Bragge. Using the Focus Group Method in Software Engineering: Obtaining Practitioner and User Experiences. In *Proceedings of the International Symposium on Empirical Software Engineering*, pages 271–280. IEEE, 2004.

[11] J. Langford and D. McDonagh. *Focus Groups: Supporting Effective Product Development*. CRC Press, 2003.

[12] L. Lehtola and S. Kujala. Requirements Prioritization Challenges in Practice. In *Proceedings of International Conference On Product Focused Software Process Improvement*, pages 497–508. Springer, 2004.

[13] A. Marchenko, P. Abrahamsson, and T. Ihme. Long-term effects of test-driven development A case study. In *Proceedings of Internation Confernce on Agile Processes in Software Engineering and Extreme Programming*, pages 13–22. Springer, 2009.

[14] M. B. Miles and A. M. Huberman. *Qualitative data analysis: An expanded sourcebook*. Sage, 1994.

[15] D. L. Morgan. Focus Groups. *Annual Review of Sociology*, 22:129–152, 1996.

[16] M. Muller and W. Tichy. Case study: extreme programming in a university environment. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 537–544, May 2001.

[17] H. Munir, M. Moayyed, and K. Petersen. Considering rigor and relevance when evaluating test driven development: A systematic review. *Information and Software Technology*, 2014.

[18] J. Nielsen. The Use and Misuse of Focus Groups. *IEEE Softw.*, 14(1):94–95, Jan. 1997.

[19] M. Pancur, M. Ciglaric, M. Trampus, and T. Vidmar. Towards empirical evaluation of test-driven development in a university environment. In *EUROCON 2003. Computer as a Tool. The IEEE Region 8*, volume 2, pages 83–86 vol.2, 2003.

[20] Y. Rafique and V. B. Misic. The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis. *Software Engineering, IEEE Transactions on*, 39(6):835–856, 2013.

[21] I. Salman, A. T. Misirli, and N. Juristo. Are Students Representatives of Professionals in Software Engineering Experiments? In *Procedings of International Conference on Software Engineering*, pages 666–676, 2015.

[22] F. Shull, G. Melnik, B. Turhan, L. Layman, M. Diep, and H. Erdogmus. What Do We Know about Test-Driven Development? *IEEE Software*, 27(6):16–19, 2010.

[23] M. Siniaalto and P. Abrahamsson. A comparative case study on the impact of test-driven development on program design and test coverage. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, pages 275–284. ACM/IEEE Computer Society, 2007.

[24] P. Tahchiev, F. Leme, V. Massol, and G. Gregory. *JUnit in action*. Manning Publications Co., 2010.

[25] B. Turhan, L. Layman, M. Diep, H. Erdogmus, and F. Shull. How effective is test-Driven Development. *Making Software: What Really Works, and Why We Believe It*, pages 207–217, 2010.

[26] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering*. Springer, 2012.