The effectiveness of test-driven development approach on software projects: A multi-case study

Vahid Bakhtiary[1], Taghi Javdani Gandomani[2], Afshin Salajegheh[3]
[1,3]Department of Computer Engineering, Tehran South University, Tehran, Iran
[2]Department of Computer Science, Shahrekord University, Shahrekord, Iran

Article Info

Article history:

Received Dec 29, 2019
Revised Mar 9, 2020
Accepted Apr 3, 2020

Keywords:

Agile software development
Code complexity
Customer satisfaction
Team productivity
Test-driven development

Corresponding Author:

Taghi Javdani Gandomani,
Department of Computer Science,
Shahrekord University,
Rahbar Blvd., Shahrekord, Iran.
Email: javdani@sku.ac.ir

ABSTRACT

Over recent years, software teams and companies have made attempts to achieve higher productivity and efficiency and get more success in the competitive market by employing proper software methods and practices. Test-driven development (TDD) is one of these practices. The literature review shows that this practice can lead to the improvement of the software development process. Existing empirical studies on TDD report different conclusions about its effects on quality and productivity. The present study tried to briefly report the results from a comparative multiple-case study of two software development projects where the effect of TDD within an industrial environment. Method: We conducted an experiment in an industrial case with 18 professionals. We measured TDD effectiveness in terms of team productivity and code quality. We also measured mood metric and cyclomatic complexity to compare our results with the literature. We have found that the test cases written for a TDD task have higher defect detection ability than test cases written for an incremental NON-TDD development task. Additionally, discovering bugs and fixing them became easier. The results obtained showed the TDD developers develop software code with a higher quality rate, and it results in increasing team productivity than NON_TDD developers.

1. INTRODUCTION

The conducted research studies show the broad acceptance of Agile methods and procedures in the software industry [1-4]. Though traditional methods are structured and sustainable in terms of nature, they have lower flexibility and compatibility and include complicated designs and heavy documentation [5]. Test-driven development (TDD) has been considered as a key practice in XP methodology [6], but can be used independently and separately. The concepts of TDD were first investigated by Beck. TDD is an evolutionary approach to development that implements a small feature in each cycle [7]. TDD is an approach, which is a combination of test-first development (TFD) and refactoring [8]. In the test-first development process, a test is first generated prior to the creation of adequate code, and then refactoring is done to convert the new code to accepted standards [9]. TDD includes 3 phases of test writing and test failing, writing the simplest code to pass the test, and the use of refactoring to eliminate iterations and achieve code clear [10]. The critical point is that TDD is not a technique to test, but it is a development process that generates more coherent codes that are less dependent on the traditional approach [11]. This process teaches programmers how to be more productive and generate codes with higher modularity, the ability to expand, and better quality [12]. Several studies have been done to investigate the effects of TDD on software development in academic and industrial environments. However, the studies which have presented documentable results with regard to productivity and quality of the developed software are scant [12]. This was the special objective of this study. The rest of the article is organized as follows: Section 2 presents the related works, and section 3 describes the research method adopted in this research. The studied cases are presented in section 4. Section 5 is assigned to the discussion, and the conclusion is presented in section 6.

2. RELATED WORKS

Since the introduction of XP, lots of papers and books have been written on TDD practice. However, relatively little research has been published on the advantages and effectiveness of TDD. A number of experimental studies have been conducted either in academic or industrial environments to investigate the effect of TDD on software quality (internal & external), test quality, and productivity. Janzen and Seiedian [13] carried out an experiment with undergraduate students in a software engineering course. Students were divided into TDD and NON-TDD groups. The results of this study show that TDD can be an appropriate approach to software design that increases code-oriented aspects such as object analysis, test coverage, and external quality, as well as developer aspects, including productivity and self-confidence.

In an academic environment, Gupta and Jalote [14] evaluated the effect of TDD on the code design, test, and quality. The results of their study indicate that TDD can be more efficient with regard to development efforts and productivity. However, the developers who have used the traditional approach had a higher level of self-confidence in code designing compared to TDD approach. In an industrial study, George and Williams [15] investigated the productivity and quality of test cases. In this survey, a total of 24 professional programmers participated. On average, the results of this survey show that 80% of professional developers consider TDD as an effective practice, and 78% claim that TDD performance causes an improvement in developers' productivity. Additionally, the results show that TDD practice leads to the creation of a simpler design, even in case of the absence of an initial design. However, the acceptance of TDD mentality is the biggest problem for some developers.

Huang and Holcombe [16] investigated the effect of TDD on effectiveness, coding effort, testing effort, and external quality. One of the results obtained from the comparative study is that TDD programmers spent a greater percentage of time for test writing and spent less time on programming. Additionally, they found that although TDD programmers have better productivity than NON-TDD, they have not necessarily delivered software with higher quality than NON-TDD. In a study conducted by Joelma et al. [17] in an academic environment on master students, it was found that most participants detected the increased quality of code, but few of them had a more comprehensive perception of the effects of TDD in designing software.

Turhan et al. [18] did an experimental research study on the effectiveness of TDD. According to this study, eight cases out of 32 experimental studies were classified as controlled experiments, of which four cases were conducted with postgraduate students. Considering the difference between the methods of study, they do not collect official results, but general findings show that quality is improved using TDD, while no effect on productivity is observed. Rafique and Misic [12] carried out a systematic analysis of 27 experimental studies that had investigated TDD impact on productivity and external quality. Eight of 27 experimental studies were classified as industry experiments. They report that quality improvement and productivity reduction in industrial studies are much higher than academic studies. Munir et al. [19] assessed the results of the systematic investigation of 41 experimental studies conducted on TDD with regard to the exact and relevant scores. In more than 19 experiments, it has been shown that there is no difference between TDD and NON-TDD given the productivity, external quality, internal code quality, number of tests, and effort expended for TDD variables. However, the authors show that there is a shortage of industrial experiments in relation to real-world systems. Table 1 shows a summary of previous studies that compare productivity and external quality for both TDD and NON-TDD approaches.

Table 1. Comparison of related works regarding TDD and NON-TDD approach

Authors-(Year)	External quality	Productivity
Scanniello et al. (2016) [20]	Better	Better
Panˇcur and Ciglariˇc (2011) [21]	No diff.	No diff.
Madeyski (2010) [22]	No diff.	No diff.
XuS (2009) [23]	Better	Better
Bhadauria (2009) [24]	-----	Better
Yenduri and Perkins (2006) [25]	Better	Better

3. RESEARCH METHODOLOGY

A case study is an experimental study, and the adoption of such studies has increased in software development environments over the past years. Some of the existing problems in these environments are very complicated, and therefore, they are affected by several factors such as processes, people, and the environment. Thus, these types of systems can provide valuable information for studying via quantitative analytical methods as a case study method [26].

3.1. Research objectives

The aim of this research is to analyze the application of TDD and NON-TDD approaches to compare them, given the external and internal quality of the work result and the productivity of developers in a software company.

3.2. Research questions

Two main questions have arisen for researchers, and answers to these questions are investigated in the following:
RQ1: Does TDD approach produce a code with higher quality than NON-TDD approach?
RQ2: Does TDD approach improve productivity?

3.3. Variables

The effectiveness of the programming approach can be investigated from different perspectives. The previous related studies have used several dependent variables and different criteria such as productivity, external quality, internal code quality, effort/time, and adaptation. Most important of them include:

- Pre-release defect: The number of defects prior to version release is calculated using $\frac{D(PRE)}{D}$ ratio. |D| is the number of defects.
- Post-release defect: $\frac{D(Post)}{D}$ the ratio is considered for the number of defects after the version release.
- Productivity: Productivity is measured based on $\frac{KLOC}{\text{Effort}}$ ratio.

Additionally, in order to calculate the mood metric, Android Studios and Visual Studio plugins were used. Each of the variables is as follows:

- MHF : Method hiding factor-encapsulation
- AHF : Attribute hiding factor-encapsulation
- AIF : Attribute inheritance factor-inheritance
- PF : Polymorphism factor-polymorphism
- CF : Coupling factor-message-passing and association

To measure the complexity of a module's decision structure, a cyclomatic complexity (v(G)) procedure is used. This is a system consisting of a number of linearly independent paths; it is the number of linearly unconventional paths and hence, in application the minimum number of paths should be verified. Essential complexity metric (ev((G)) is used to measure the degree of unstructured constructs in a module. this metric system evaluates the quality of the codes and their structures, used to predict maintenance effort and to help with the modularization process. Module design complexity (iv(G)) is used to measure the complexity of a design-reduced module, reflecting the complexity of calling patterns onto its immediate subordinate parts. This metric system differentiates between modules, resulting in serious complications in the design of any program they are a part of, and any modules that simply contains complex computational logic. This is the basis upon which integration complexities (S0 and S1) and program design are deliberated.

4. DESCRIPTION OF THE STUDIED CASES

This is a multiple-case study related to software development teams. In this part, two cases were selected for this study and were investigated. The initial purpose of this case study is the development of a system by two different teams of experienced persons. Therefore, two different projects were considered, which have been numbered with alphabetical letters.

4.1. Case A

Case A is an android application (app) in the java programming language. This app has been designed to present healthcare services to the public, especially patients. Doctors and healthcare centers can be provided with all of the electronic health services through this app and can manage their patients. The information and patients in each center are only accessible to the same center. A team performed the development stages using TDD perspective, while the other team did not pay attention to this practice. Overall, four persons participated in this project in different roles, such as developer and project manager.

The total duration of this project was three months. In the development stage, the determined requirements are implemented repeatedly. The requirements have been organized with regard to the app use cases. Testing activities in the development stage, including unit test and integration tests, were the responsibility of the developers one by one. In the development stage, the system test was conducted only informally to present the initial feedback. Additionally, system and regression test based on the determined test items were the main activity in the stabilization phase, together with resolving the discovered problems. After the final phase, the output of each team was published in the form of an app, including the new characteristics, and each app was delivered to a therapeutic center. The studied project description is presented in Table 2.

Table 2. Assessment results, case A

Variables	TDD	NON-TDD
Duration (hour)	360	360
Team effort (hour)	345	355
Team size	4	4
Experience level (<5 years, 6-10 years, >10 years)	3-5	3-5
Applied technology	Android studio-Java	Android studio-Java
customer satisfaction	95%	90%
Source LOC	29256	29680
Class coupling average	21.93%	56.67%
Pre-release defect	12	31
Post-release defect	11	23
Development productivity	84.4	83.6
Test coverage	88%	67%

Data collected from the comparison of two approaches show that the number of discovered defects before and after the publication is far better than NON-TDD, which has caused an increase in customer satisfaction. It is worth mentioning that the level of customer satisfaction was directly asked from the customer in each cycle, and the mentioned number is the mean of all cycles. Moreover, productivity is nearly the same in both approaches, but TDD approach has performed a little better. Figure 1 shows the results of productivity and satisfaction.

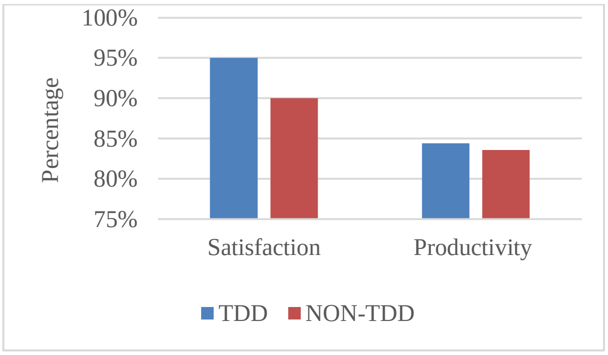


Figure 1. Results of productivity and satisfaction, case A

Result shows the dependence between classes in TDD is far less than NON-TDD method; thus, it can be concluded that modularity is higher in TDD. Cyclomatic complexity is, in fact, the number of independent paths in the code. The more the branches and paths, the more complicated it is. As shown in Table 3, TDD approach was a great help to the reduction of complexity. Table 4 shows the results of evaluating two approaches based on the design metrics.

Table 3. Results of complexity assessment, case A

Approaches	ev(G)	iv(G)	v(G)
TDD	1.08	1.18	1.24
NON-TDD	4.45	6.39	6.40

Table 4. Results of design assessment, case A

Variables	TDD	NON-TDD
AHF	90.38%	52.39%
AIF	89.13%	94.13%
CF	21.93%	56.67%
MHF	4.64%	12.88%
PF	128.57%	100%

According to the obtained results, the quality of design in TDD approach has outperformed NON-TDD approach in most of the variables, indicating the better design quality in TDD. In order to better understand the results of cyclomatic complexity, are shown graphically in Figure 2. Figure 3 shows the results as a diagram.

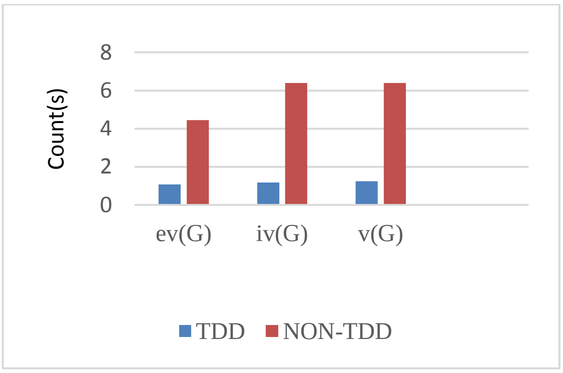


Figure 2. Complexity assessment graph, case A

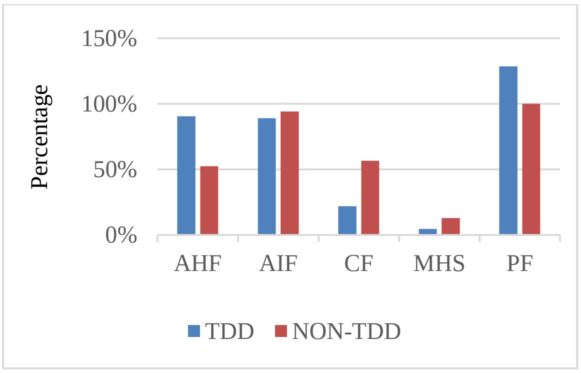


Figure 3. Design assessment graph, case A

4.2. Case B

Case B is the creation of a web-based customer relationship management (CRM) portal. This software can be implemented on all versions of browsers such as Internet Explorer, Firefox, Safari, and Chrome. This software is multilingual and can be used internationally. Various programming technologies and languages (C#, JavaScript, CSS, Ajax) have been used to create the given system. To this purpose, two teams of five were investigated. One of the teams followed TDD approach, while the other team followed NON-TDD. It took six months to conduct the case study. The team members composed of experienced individuals and students in their final year of bachelor's studies. The studied project description is presented in Table 5.

Table 5. Assessment results, case B

Variables	TDD	NON-TDD
Duration (hour)	700	700
Team effort (hour)	650	665
Team size	5	5
Experience level (<5 years, 6-10 years, >10 years)	3-5	3-5
Applied technology	Visual studio-C#	Visual studio-C#
customer satisfaction	95%	89%
Source LOC	51350	52003
Class coupling average	22%	67.11%
Pre-release defect	16	42
Post-release defect	10	27
Development productivity	79	78.2
Test coverage	86%	61%

The obtained results show that by increasing the project scale, the number of discovered defects before and after release more increased in the NON-TDD approach compared to TDD. This has reduced customer satisfaction in this method compared to TDD approach. It is worth mentioning that the level of customer satisfaction was directly asked from the customer in each cycle, and the mentioned number is the mean of all cycles. Moreover, productivity is nearly the same in both approaches, but TDD approach has performed a little better. Figure 4 shows the results of productivity and satisfaction. The obtained results in Table 6 for measuring cyclomatic complexity show that TDD approach has reduced the complexity of branches and paths available in code compared to NON-TDD approach. Therefore, it helps with easier maintenance. Table 7 shows the results of evaluating two approaches based on the mood metric. In order to better understand, Figure 5 present the cyclomatic complexity results in graphically. According to the obtained results, it can be concluded that TDD approach has outperformed NON-TDD approach, indicating the better quality of design in TDD. Figure 6 shows the results as a diagram.

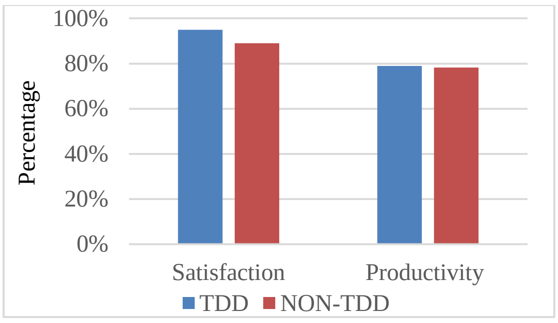


Figure 4. Results of productivity and satisfaction, case B

Table 6. Results of complexity assessment, case B

Approaches	ev(G)	iv(G)	v(G)
TDD	1.48	1.58	2.0
NON-TDD	5.1	7.45	7.45

Table 7. Results of design assessment, case B

Variables	TDD	NON-TDD
AHF	85.11%	45.79%
AIF	78.41%	83.22%
CF	23.62%	75.44%
MHF	12.33%	19.56%
PF	108%	95%

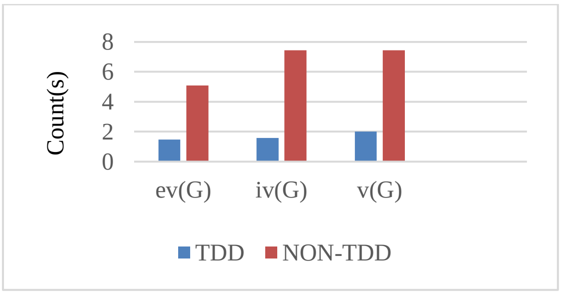


Figure 5. Complexity assessment graph, case B

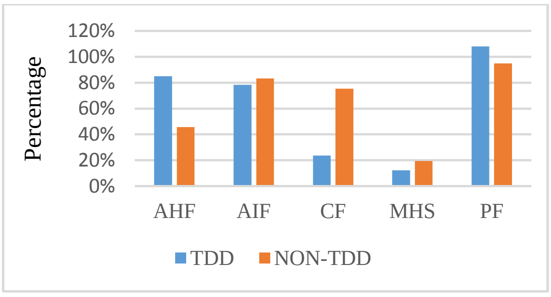


Figure 6. Design assessment graph, case B

5. DISCUSSION

In this section, the obtained results with regard to RQs are discussed and the possible practical concepts from the research are presented. TDD needs a new way of thinking because some methods such as the first test and the continuous improvement of design are developed through reforming which is not common among developers. Additionally, research studies show that it is difficult for programmers to accept the TDD mentality [27]. One of the challenges ahead of accepting TDD by developers is the time-consuming process of test writing at the beginning of the project, and this reduces productivity. For them, the major problem, at first, is to know where to start and then to know how to make a test for a feature that does not still exist. But the results of the research show that this time is compensated for in the following due to spending less time to find bugs and resolving them.

TDD approach is started by writing tests, which indicate the functional requirements and the developers should write the simplest code to pass them. This persuades developers to develop software through small steps [28]. Therefore, TDD can be an alternative to incremental test-last and can be as effective as it was, provided that granularity is performed. The dependence between classes shows that this modularity has been complied with to a relatively good extent, which can increase the reusability. Data analysis shows that in response to RQ1, TDD approach leads to producing software products with higher quality. Generally, no critical situation was observed in design quality. Additionally, it seems that as time passes and more tests are added, the change management also becomes easier during the use of TDD. In response to RQ2, it can be said that there are few experimental evidence showing that TDD causes an increase or a decrease in productivity [21]. The results of the research showed that the TDD has improved effectiveness to some extent compared to NON_TDD, but most developers do not feel it when they use TDD for the first time.

However, at first, developers may find it difficult to use TDD approach, especially for inexperienced developers, because they want to implement the requirements as soon as possible. On the other hand, the experienced developers know that TDD stages can help reduce design complexity. According to the results, it can be concluded that TDD approach increases the developers' perception of the system requirements because before they write tests, they should think about how characteristics function. In addition, they can use test cases to explain their code, which helps with their perception. Therefore, the results show that TDD approach causes an increase in code quality. Thus its legibility and simplicity increase for developers. Moreover, the software maintainability increases because writing tests before code compels developers to decide to make a better design during development. Additionally, discovering bugs and fixing them becomes easier. According to Turhan et al. [18], it is expected that the use of TDD approach and incremental approach causes a simple and straightforward design to be created.

5.1. Implications for theory

The main issue focused on this study was how well TDD can affect the software development process in terms of code quality and team productivity. TDD is usually known as effective and modern testing techniques that impact the design process in advance. The rationale behind the usefulness of this Agile practice has mainly root in focusing on the user acceptance criteria. The result of the study showed empirical results regarding the importance and effectiveness of TDD. Most often, software teams are encouraged to employ TDD, and this study showed the excellence of this approach in practice compared to NON-TDD approach. It seems that this practice can lead to achieving software code with higher quality in practice. Also, since TDD may lead to fewer errors and re-works, it can also indirectly increase the productivity of development teams. This fact is also supported by the results of the study.

6. CONCLUSION

This case study was conducted with the aim of assessing TDD in software development. In order to investigate the effect of TDD approach compared to NON-TDD, the effects of this approach on productivity and software quality were investigated. Results show that TDD programmers write more tests leading to higher levels of productivity and can improve productivity. Moreover, advancing a test development at a time and writing test before implementation help with better analysis and improve the required perception of the main requirements and reduce the scope of task performance. In addition, the small range of tests and rapid iteration, which is possible by regression test, reduce debugging and violations. The obtained results show that the design quality is more desirable in using TDD approach than NON-TDD. Overall, TDD effectiveness can affect developers' encouragement to use it in projects. Considering the limitations, it should be considered that the number of participants and the duration of project implementation may be inadequate to generalize the results. Additionally, further research needs to be done on larger projects.

REFERENCES

[1] M. Majchrzak and L. Stilger, “Experience report: Introducing Kanban into automotive software project,” *e-Informatica Software Engineering Journal,* vol. 11, no. 1, pp. 39-57, 2017.

[2] B. Kitchenham, L. Madeyski, D. Keung, P. Brereton, S. M. Charters, S. F. Gibbs, and A. Pohthong, “Robust statistical methods for empirical software engineering,” *Empirical Software Engineering,* vol. 21, no. 1, pp. 579-630, 2017.

[3] L. Madeyski and M. Kawalerowicz, “Software engineering needs Agile experimentation: A new practice and supporting tool,” in *Software Engineering: Challenges and Solutions* (L. Madeyski, M. Smialek, B. Hnatkowska, and Z. Huzar, eds., vol. 504 of *Advances in Intelligent Systems and Computing,* pp. 149-162, 2017.

[4] L. Madeyski and M. Kawalerowicz, “Continuous test-driven development-a novel Agile software development practice and supporting tool,” in *Proc. of the 8th Int. Conf. on Evaluation of Novel Approaches to Software Engineering (ENASE),* pp. 260-267, 2013.

[5] V. Aggarwal and A. Singhal, “Empirical study of test driven development with scrum,” in *International Conference on Advances in Computing and Data Sciences, Springer,* pp. 13-21, 2019.

[6] A. Tosun, M. Ahmed, B. Turhan, and N. Juristo, “On the effectiveness of unit tests in test-driven development,” in *Proceedings of the 2018 International Conference on Software and System Process,* pp. 113-122, 2018.

[7] N. C. Borle, M. Feghhi, E. Stroulia, R. Greiner, and A. Hindle, “Analyzing the effects of test driven development in GitHub,” *Empirical Software Engineering,* vol. 23, no. 4, pp. 1931-1958, 2018.

[8] E. Guerra and M. Aniche, “Achieving quality on software design through test-driven development,” *Software Quality Assurance-Chapter 9,* pp. 201-220, 2016.

[9] K. Bajaj, H. Patel, and J. Patel, “Evolutionary software development using test driven approach,” *2015 International Conference and Workshop on Computing and Communication (IEMCON),* pp. 1-6, 2015.

[10] S. Romano, D. Fucci, M. T. Baldassarre, D. Caivano, and G. Scanniello, “An empirical assessment on affective reactions of novice developers when applying test-driven development,” *International Conf. on Product-Focused Software Process Improvement,* pp. 3-19, 2019.

[11] D. Janzen and H. Saiedian, “Does test-driven development really improve software design quality?,” in *IEEE Software,* vol. 25, no. 2, pp. 77-84, 2008.

[12] Y. Rafique and V. B. Mišić, "The effects of test-driven development on external quality and productivity: A meta-analysis," *IEEE Transactions on Software Engineering,* vol. 39, no. 6, pp. 835-856, 2013.

[13] D. S. Janzen and H. Saiedian, “On the influence of test-driven development on software design,” in *19th Conference on Software Engineering Education & Training (CSEET'06),* pp. 141-148, 2006.

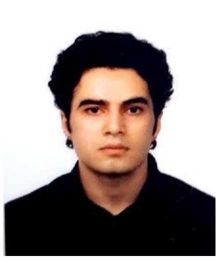
[14] A. Gupta and P. Jalote, “An experimental evaluation of the effectiveness and efficiency of the test driven development,” in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007),* pp. 285-294, 2007.

[15] B. George and L. Williams, “A structured experiment of test-driven development,” *Information and software Technology,* vol. 46, no. 5, pp. 337-342, 2004.

[16] L. Huang and M. Holcombe, “Empirical investigation towards the effectiveness of test first programming,” *Information and Software Technology,* vol. 51, no. 1, pp. 182-194, 2009.

[17] J. Choma, E. M. Guerra, and T. S. da Silva, "Developers' initial perceptions on TDD practice: A thematic analysis with distinct domains and languages," *International Conference on Agile Software Development*, pp. 68-85, 2018.
[18] 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-Chapter 12*, pp. 207-217, 2010.
[19] H. Munir, M. Moayyed, and K. Petersen, "Considering rigor and relevance when evaluating test driven development: A systematic review," *Information and Software Technology*, vol. 56, no. 4, pp. 375-394, 2014.
[20] G. Scanniello, S. Romano, D. Fucci, B. Turhan, and N. Juristo, "Students' and professionals' perceptions of test-driven development: a focus group study," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pp. 1422-1427, 2016.
[21] M. Pančur and M. Ciglarič, "Impact of test-driven development on productivity, code and tests: A controlled experiment," *Information and Software Technology*, vol. 53, no. 6, pp. 557-573, 2011.
[22] L. Madeyski, "Test-driven development: An empirical evaluation of Agile practice," *Springer Science & Business Media*, 2009.
[23] S. Xu and T. Li, "Evaluation of test-driven development: An academic case study," in *Software Engineering Research, Management and Applications 2009*, pp. 229-238, 2009.
[24] V. S. Bhadauria, "To test before or to test after-an experimental investigation of the impact of test driven development," Doctoral Thesis, The University of Texas at Arlington, 2009.
[25] S. Yenduri and A. L. Perkins, "Impact of using test-driven development: A case study," *Software Engineering Research and Practice*, vol. 1, no. 2006, pp. 126-129, 2006.
[26] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131-164, 2009.
[27] J. Garbajosa, X. Wang, and A. Aguiar, "Agile processes in software engineering and extreme programming," *Springer International Publishing*, 2018.
[28] D. Fucci, H. Erdogmus, B. Turhan, M. Oivo, and N. Juristo, "A dissection of the test-driven development process: Does it really matter to test-first or to test-last?," in *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 597-614, 2016.

BIOGRAPHIES OF AUTHORS



Vahid Bakhtiary received BS in Software Engineering from Islamic Azad University. He received MS in Software Engineering from Islamic Azad University. Currently, he is Pursuing PhD degree at Software Engineering in Faculty of Software Engineering, IAU. His research interest includes Software engineering, Software architecture, and Agile methodology.

Taghi Javdani Gandomani received his Ph.D. degree in software engineering from Universiti Putra Malaysia (UPM), Malaysia, in 2014. He currently serves as an Assistant Professor in Shahrekord University, Shahrekord, Iran. His research interests include software methodologies, Agile methods, software processes, and software metrics.

Afshin Salajegheh has received his BS from Tehran University and MS in Artificial Intelligence and PhD in software engineering from Islamic Azad university science and research branch. He is a faculty member of software engineering and computer science at the Islamic Azad University South Tehran Branch since 1998. His major interests are software engineering, software architecture, Data mining, and Database. He also has worked as a Senior IT Consultant, system analyst and Designer, Programmer, Project Manager, and Data scientist for more than 24 years.