Information and Software Technology 74 (2016) 45–54

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

The effects of test driven development on internal quality, external quality and productivity: A systematic review

Wilson Bissi[a,*], Adolfo Gustavo Serra Seca Neto[b], Maria Claudia Figueiredo Pereira Emer[b]

[a] Information Technology Department, Global Village Telecom, 2197 Dário Lopes dos Santos Avenue, Curitiba, Paraná, 80210-010, Brazil
[b] Academic Department of Informatics, Federal University of Technology - Paraná, 3165 Sete de Setembro Avenue, Curitiba, Paraná, 80230-901, Brazil

ARTICLE INFO

Article history:
Received 2 December 2015
Revised 12 February 2016
Accepted 15 February 2016
Available online 24 February 2016

Keywords:
Test-driven development
Productivity
Internal quality
External quality
Systematic review

ABSTRACT

Context: Test Driven Development (TDD) is an agile practice that has gained popularity when it was defined as a fundamental part in eXtreme Programming (XP).

Objective: This study analyzed the conclusions of previously published articles on the effects of TDD on internal and external software quality and productivity, comparing TDD with Test Last Development (TLD).

Method: In this study, a systematic literature review has been conducted considering articles published between 1999 and 2014.

Results: In about 57% of the analyzed studies, the results were validated through experiments and in 32% of them, validation was performed through a case study. The results of this analysis show that 76% of the studies have identified a significant increase in internal software quality while 88% of the studies identified a meaningful increase in external software quality. There was an increase in productivity in the academic environment, while in the industrial scenario there was a decrease in productivity. Overall, about 44% of the studies indicated lower productivity when using TDD compared to TLD.

Conclusion: According to our findings, TDD yields more benefits than TLD for internal and external software quality, but it results in lower developer productivity than TLD.

1. Introduction

According to the annual report conducted by Version One [1], Test Driven Development (TDD) is one of the agile practices most frequently used in the software development industry. About 38% of respondents answered that they use this practice. A study conducted in Brazil showed that 39.50% of respondents use TDD for software development [2].

According to Beck [3], TDD is a software development practice where automated unit tests are incrementally written even before the source code is implemented.

TDD has gained popularity when it was defined as a fundamental part of the development process called Extreme Programming (XP) [4]. Currently, this practice is used independently in the software industry.

Several studies have focused on the effects produced by TDD practice on software development within the academic setting (universities) or in the industry (companies), comparing TDD with Test Last Development (TLD). However, most of these studies failed to provide conclusive results with regard to productivity and quality of developed software [5].

The present paper selected and classified academic papers that analyzed the effects of TDD on productivity, internal quality and external quality of software design, when compared with the TLD practice. Only papers that were published between 1999 and 2014 were selected for this purpose.

The process of systematic review used in this study initially identified 1107 papers; 27 of them were selected and discussed in detail. By reviewing each selected study in detail, this paper provides greater emphasis on variables used in research validation; for example, the setting, the language used in development, the profile of participants, the type of research method applied and the findings of each study.

After being consolidated, the findings of the present study showed that in about 57.14% of the analyzed studies, the results were validated through experiments, and in 32.14% of the studies, validation was performed through a case study.

In addition, the results of the analysis of these papers showed that for 76% of the studies, there was a significant increase in internal software quality and there was a significant increase in

* Corresponding author. Tel.: +55 4499131197.
E-mail addresses: wbissi@gmail.com, wilson_bissi@hotmail.com (W. Bissi), adolfo@utfpr.edu.br (A.G. Serra Seca Neto), mcemer@utfpr.edu.br (M.C.F.P. Emer).

http://dx.doi.org/10.1016/j.infsof.2016.02.004
0950-5849/

external software quality in 88% of them. With regard to productivity, there was an increase in the academic environment, while in the industrial environment, productivity was decrease. Overall, about 44% of the studies indicated lower productivity when the TDD practice was used, compared with TLD.

The remaining sections of this paper are organized as follows: Section 2 is focused on describing the TDD practice; Section 3 has a detailed description of the research method and the studies selected for this review. Section 4 presents the results of the analyses. Section 5 reports the threats to the validity of this systematic review. Section 6 discusses the findings and Section 7 presents related works to this systematic review. Finally, Section 8 presents the conclusion and suggestions for further research.

2. Context

Test Driven Development is a software development practice whose core idea is to design software incrementally by conducting unit tests that will guide the development process. Firstly, developers identify the features and write the corresponding unit tests to express the desired functionality [6]. The functionality will be implemented only after the unit test has been written.

A unit test examines the behavior of a distinct unit of work [7]. This level of testing should to ensure that the smaller units (module, class or method) are operating in accordance with what was specified, independent of the rest of the system.

Kent Beck [3] defines TDD as a set of techniques that encourage the development of simple projects and the development of a test suite. According to Beck [8], although TDD is focused on creating automated unit testing, it is not exactly a testing technique. It should be considered as a software design technique [9].

In the TDD cycle, the test and the code implemented are usually related to a small unit of software, a method or a function. Therefore, the tests written in this cycle are unit tests.

TDD is also known by the red-green-refactor cycle [3], which consists of the following steps:

1. Design and add a unit test;
2. Run all tests and check the failure of the new test added in step 1 (red);
3. Add new code that is sufficient to satisfy the new test;
4. Run all tests, repeat step 3 if necessary until all tests have passed (green);
5. Refactor to improve the code/test structure;
6. Run all the tests after refactoring to ensure that all tests have passed.

As noted in the red-green-refactor cycle, before implementing a new feature, a unit test should be written, and only after it fails, the feature code must be developed. At the end of the red-green-refactor cycle, the developer should refactor the code and tests before starting the development of the next feature. This refactoring is required to improve the internal structures of the software.

The comparison between TDD and TLD is the basis for many previous studies. Fig. 1 shows and compares the development flow followed in each of the practices, and clarifies the differences between them.

Fig. 1 shows the development of a new functionality in TDD and TLD flows. The main difference shown in Fig. 1 is that in the TDD flow, the second step is to write the test and then run it; if it fails, the functionality source code is written to pass test and if necessary, the source code is refactored. While in the TLD flow, the second step is to write the functionality source code and then the test is written and performed [11].

In TLD, developers can write tests iteratively after the completion of the feature code or choose to write all the tests at the end of the implementation of the entire system. This depends on the development process model specified for the software development. However, in TLD a unit test should be written only after the feature code has been finalized.

Table 1
Online libraries.

Source	Search date
IEEE Xplore	18/12/2014
ScienceDirect	18/12/2014
ACM Digital Library	19/12/2014
CiteSeerx	20/12/2014
Wiley Online Library	20/12/2014

3. Method

The research method used in this study was a systematic review of the literature, following the guidelines described in [12]. A systematic review is an empirical study in which a research question or hypothesis is addressed to gather evidence of a number of primary studies through a systematic process of research and data extraction [13].

The search and selection phases of this systematic review ware conducted by the first author who holds undergraduate and specialization degrees in software development and currently is a graduate student in applied computing.

3.1. Research process

The research process of the systematic review was started by defining the research protocol, which defines the purpose of the review. As a first task, four research questions were formulated.

Defining the research questions or hypotheses is an essential part of the systematic review because they will guide the entire review [14].

1. What are the main effects achieved by applying the TDD practice to software development?
2. Which development paradigms is the TDD practice applied to?
3. How does the practice of TDD influence productivity as well as internal and external software quality?
4. What are the effects of the TDD practice as measured during the software development process?

After the research questions were defined, the primary studies were identified in the knowledge bases listed in Table 1 by means of the research string defined in Section 3.2. After the search was performed in the knowledge bases, the works were selected and classified as shown in Section 3.3. Section 3.4 presents the criteria used to select the papers, and Section 3.5 describes how the data were extracted from the analyzed papers.

3.2. Identification of primary studies

In order to find the relevant studies, the most important search terms were identified and selected. This review adopted a guideline widely used in the medical field to identify the effectiveness of a treatment, which implies the use of three points of view: Population, Intervention and Outcomes.

These guidelines were initially presented by [15] and extended later in [16] with the following definition:

- *Population:* any specific role of software engineering or the field of application of a research study.
- *Intervention:* software technologies that address specific issues.
- *Results:* relative to factors of importance to researchers.

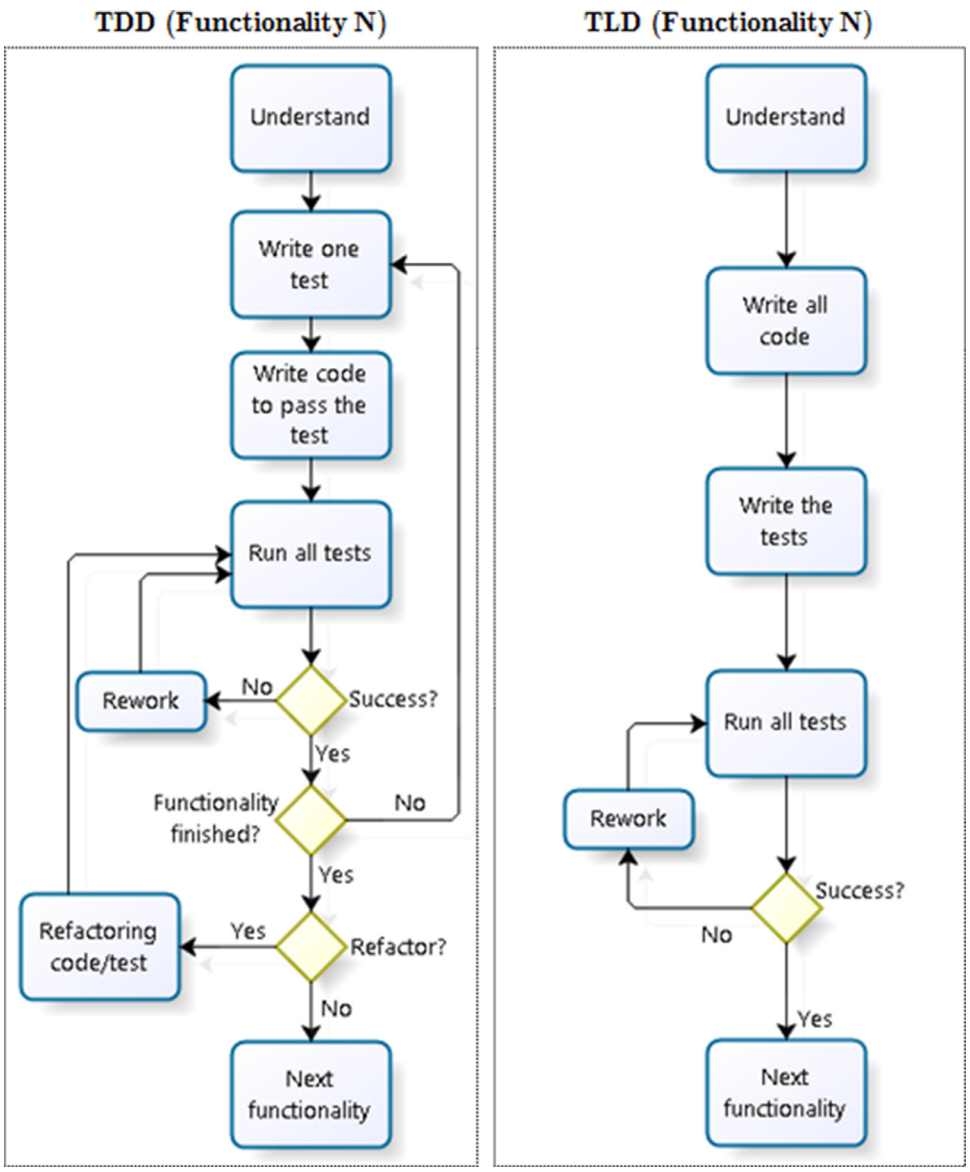

Fig. 1. Differences between TDD and TLD flows, based on [10].

The terms were defined for each of the three points of view, as follows:

- *Population:* Software Engineers, Software Developers, Programmers.
- *Intervention:* Test Driven Development, Test-Driven Development, TDD, Test First Development.
- *Results:* Code Quality, Quality Improvement, Design Improvement, Improved Software Development, Internal Quality, External Quality, Productivity.

The construction of the search string and the logical notation for the search are defined as follows:

$$(P_1 \text{ OR } P_2, \ldots, \text{OR } P_n) \text{ AND } (I_1 \text{ OR } I_2, \ldots, \text{OR } I_n)$$

where P_n refers to the Population terms and i_n refers to the Intervention terms. They are related by the Boolean operators AND and OR. To expand the search results, only the Population and Intervention terms were considered. The inclusion of the Results field on the search considerably reduces the volume of relevant studies identified and, thus, increases the risk of the absence of these studies in the initial search [17].

The resulting search string was used to perform the search in five online libraries shown in Table 1:

During the search in the afore-mentioned online libraries, some changes were required to meet the configuration of the search criteria in each site. The search for papers was restricted to the range of years of publication (1999 to 2014) in English. This range was chosen because it was assumed that the term TDD started to be used more often after the advent of agile methods, particularly the XP method, which emerged in 1999 as suggested by [4]. The fields considered in the search for papers were: Abstract, Title and Keywords (TITLE-ABS-KEY).

3.3. Paper selection process

The papers were selected in four filtering stages until the final set of primary studies that was analyzed. Fig. 2 shows the details of this process.

Initially, 1107 papers were collected with the search string performed in the knowledge bases, as shown in Table 1. Then, 143 duplicate papers were excluded. In the next phase, seven papers were excluded because there was no full access to the full text of the published material. Following the exclusion criteria (see Section 3.4), other 893 papers were excluded after their title and abstract had been read, thus, 64 papers remained for analysis. Most

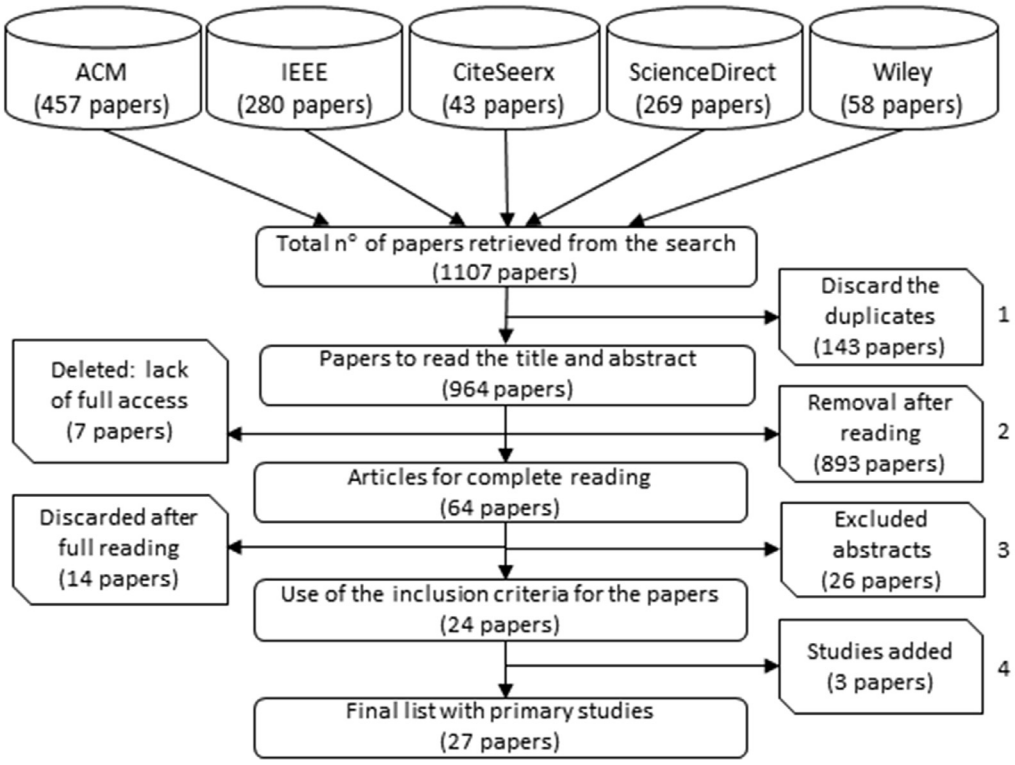

Fig. 2. Paper selection process.

of the articles removed in this step neither matched inclusion criteria 3 and/or 4 (Section 3.4) or were not experiments. Out of this total, Twenty-six papers that had less than 4 pages were also excluded, because they were abstracts. In addition to the abstracts, 14 papers were excluded after full reading, because of the exclusion criteria 1, 2 and 3 described in Section 3.4.

To complement the list of papers, the snowball sampling technique [18] was used. This technique aims to analyze paper references, particularly systematic reviews, in order to find relevant studies that had not been selected during the search in the knowledge bases. While reading the references of the papers presented in Section 7, three new items were included to the final set, consolidating a total of 27 selected papers.

To identify and assess the quality of the studies gathered in the selection of papers, seven validation questions (VQ) were developed, based on the definitions of [14] and [19]. These questions were divided into three groups: Design, Analysis and Conclusion. The Design group contains questions 1 and 2, the Analysis group contains questions 3 and 4, and the Conclusion group contains the last three questions: 5, 6 and 7. The questions are listed below.

1. Are the techniques related to the research questions that this study aims to answer?
2. Were the metrics used in the studies completely defined?
3. Is there any reliable technical validation of the approach?
4. Were participants in the study or the observation unit clearly defined?
5. Are the results explained and associated with the objectives of the study?
6. Were the results validated? Can they be replicated?
7. Were the limitations of the study properly explained?

The papers included in this review have satisfied all these requirements. This validation is required to increase the quality and the reliability of the review.

3.4. Criteria for inclusion and exclusion of papers

Five conditions (listed below) were defined for inclusion of the papers found in the search in the digital libraries (Table 1). A paper was only included if it met all the conditions below and did not violate any exclusion criteria:

1. Papers must be fully available;
2. Papers must have been previously published in any scientific format (journal or conference proceedings);
3. Papers must present results on the TDD practice in software development;
4. Papers need to adequately describe the research context in which the study was conducted;
5. The results must be based on a clear measurement criterion.

In addition to the inclusion criteria, six exclusion criteria were defined. The papers were thus excluded from the process of systematic review if they satisfied any of the criteria below:

1. Duplicate papers;
2. Papers that are not related to the TDD practice in software development;
3. Systematic reviews of the TDD practice;
4. Papers that were not written in English;
5. Papers that were not published in an academic format;
6. Papers that are not fully accessible.

The purpose of these criteria is to keep focus while reading the papers and make more informed decisions about the selection of the studies.

3.5. Data extraction process

In order to extract information from the studies to be analyzed in this systematic review, a list of attributes was created as a

Table 2
Details of information extraction from the papers.

Category	Attributes
Information	Title, university, authors, year, publication format
Setting	Academic or industrial
Method	Case study, experiment, questionnaire, simulation, mixed
Domain	desktop, web, embedded, distributed
Language	Java, PHP, C#, etc.
Methodology	Waterfall, agile, etc.
Project size	Size measured in units of time or lines of code
Number of people	Number of people that worked in the project
Experience	Beginner (up to 2 years), intermediate experience (3–7 years), experienced (over 7 years)
Focus of studies	Internal quality, external quality and productivity
Findings	Advantages and disadvantages of TDD

Table 3
Empirical studies on TDD.

Paper	Year	Setting	Language	Participants
[20]	2007	Industrial	NI[a]	Professionals
[21]	2004	Academic	NI	Professionals
[22]	2004	Industrial	Java	Professionals
[23]	2007	Academic	Java	Students
[24]	2003	Industrial	Java	Professionals
[25]	2003	Industrial	Java	Professionals
[26]	2008	Mixed	Java	Mixed
[27]	2009	Academic	Java, ActionScript	Students
[28]	2003	Academic	Java	Students
[29]	2007	Academic	Java, C++	Students
[30]	2004	Academic	Java	Students
[31]	2008	Academic	C++	Students
[32]	2006	Industrial	Java	Professionals
[33]	2006	Industrial	C, C#, C++	Professionals
[34]	2006	Industrial	NI	Students
[35]	2006	Academic	Java	Students
[36]	2005	Academic	Java	Students
[37]	2007	Industrial	Java	Professionals
[38]	2012	Industrial	Java	Professionals
[39]	2008	Industrial	Java	Professionals
[10]	2011	Academic	Java	Students
[40]	2010	Mixed	NI	Mixed
[41]	2006	Industrial	Java, C++	Professionals
[42]	2010	Academic	Java	Students
[43]	2007	Academic	Java	Students
[44]	2008	Industrial	Java, C++, .NET	Professionals
[45]	2006	Academic	Java	Students

[a] NI: Non-informed.

model to consolidate and organize information for all the papers. Table 2 shows the defined attributes.

The attributes shown in Table 2 extract general and specific information about the papers as regards methodologies, project size, research method, technologies in use, implementation of the study, etc.

Based on these attributes, the extracted data were grouped and described in detail, thus facilitating the analysis and comparison reported in Section 4.

4. Analysis and results

A total of 27 studies were selected and analyzed in this systematic review. These studies were published between 1999 and 2014, and they make a comparison between TDD and TLD practices. It is in this comparison that the effects are identified and described in detail.

Table 3 lists all the papers found and selected for this review. This table also describes the year of publication, setting in which the studies were applied, the programming language used in the study, and the profile of the participants (professionals or students).

Table 4
Mapping of research methods used in the studies.

Method	Studies	N° Studies
Experiment	[10,21–32,35,36,42]	16
Case Study	[26,33,37,39,41,43–46]	9
Questionnaire	[38,40]	2
Simulation	[34]	1

As can be seen in Table 3, the setting in which the studies were applied were divided into two types: Academic (48.14%) and Industrial (44.46%). Two other studies (7.40% of the total) used the two scenarios and for this reason, they were classified as Mixed.

As far as programming languages are concerned, virtually all studies used either Java or C++ to implement the work. Most of the studies (23 of them) were published before 2010.

The profile of the participants of the studies matches the same proportion of the settings. There are two types of participants: Students and Professionals, each present in 44.46% and 48.14% of the papers, respectively. Two other studies, or 7.40% of the total, have used the two types of participants and were thus classified as Mixed.

Although the works were performed in the industrial environment with professionals, not all studies were conducted using real projects. Only 9 out of 13 works conducted in the industrial environment used real-world projects, while the others made use of fictitious exercises in order to validate the study.

Another classification was carried out for the applied method of study. The four research methods found in the publications were: Experiment (57.14%), Case Study (32.14%), Questionnaire (7.14%) and Simulation (3.58%.)

The years of publication considered for the search of papers ranged from 1999 to 2014. Fig. 3 shows the distribution of published papers per year. Within the scope of the present review, the first papers were published in 2003 (three publications); by the year 2006, that number had doubled. Further research studies were conducted in the following years, but less often than in 2006.

The study conducted in [26] shows two validation methods: one was an experiment and the other, a case study. This study was counted twice, once in each method. Therefore, the sum of the number of papers shown in Table 4 is larger than the total number of selected papers in this review.

Table 4 shows a consolidated list of the research methods found in the studies. The most frequent method was the experiment (16 papers), followed by the case study (9 papers). Only two papers used the questionnaire as a method and only one study used simulation to conduct research.

The categorization of primary studies was followed by observation of their effects, which are described in Section 4.1.

4.1. Observation of effects in the primary studies

This section presents the results of the observations made in the studies found in this systematic review. The effects reported in this section are relative to the comparison between TDD and TLD.

In the following tables, some works are repeated in more than one classification because they used more than one setting or method for applying the study, thus, they are considered independent studies. The papers [26] and [40] are studies with mixed settings (academic and industrial). The paper [26] used the experiment and the case study as a method for the application of the study.

The results reported in this section cannot be thoroughly compared because the metrics used by the studies are not uniform

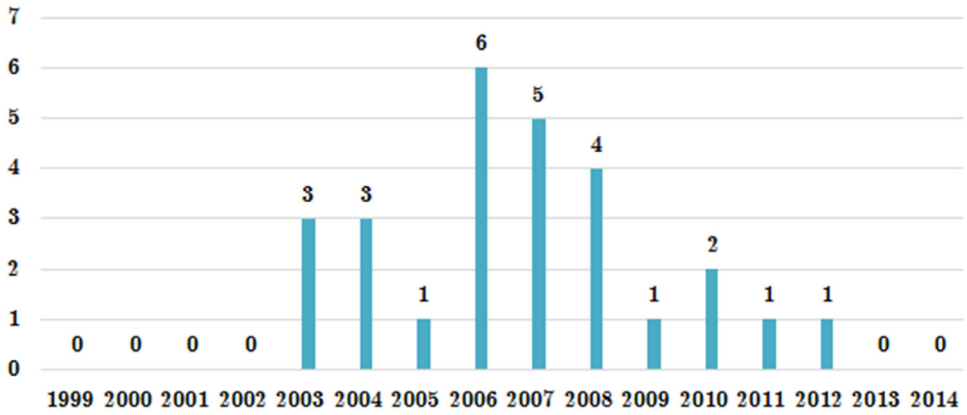

Fig. 3. Distribution of papers by year of publication.

Table 5
Studies that show the effects of TDD on internal software quality.

Method	Setting	(+) IQ	(=) IQ	(−) IQ
Experiment	Academic	[10,21,26,27,29,36]	[31,42]	[28,35]
Experiment	Industrial	[22,24–26,32]		
Case Study	Academic	[26,45]		
Case Study	Industrial	[20,26,33,37]		
Questionnaire	Academic	[40]		
Questionnaire	Industrial	[40]	[38]	
Simulation	Industrial	[34]		

Table 6
Studies that show the effects of TDD on external software quality.

Method	Setting	(+) EQ	(=) EQ	(−) EQ
Experiment	Academic	[10,21,27,30]	[23]	[28]
Experiment	Industrial	[22,24,25]		
Case Study	Academic	[45]		
Case Study	Industrial	[33,39,41,44]		
Questionnaire	Academic	[40]		
Questionnaire	Industrial	[40]		
Simulation	Industrial	[34]		

or standardized. However, the evidence was summarized and presented in the following tables, based on the metrics reported in each of the analyzed studies.

4.2. Internal quality

Table 5 lists the studies and the effects found for internal software quality. The main indicator used to identify the effects was the test coverage in the resulting production code. It is known that, when analyzed separately, this indicator does not guarantee that the code has a high level of internal quality. As the test coverage metric was the only metric common to all selected papers, thus, the comparison between papers used only this metric.

The code coverage may indicate that the software has some flaws. However, the form of measurement and the results of the analyzed studies were not questioned in this review.

In Table 5, the papers contained in the column **(+) IQ** are those that showed positive results and increased internal software quality when the TDD practice was applied. The column **(=) IQ** shows papers that did not identify any effect when applying the two practices, while the column **(−) IQ** contains the papers where negative effects were found for internal quality by applying the TDD practice, i.e., internal quality decreased.

Table 5 shows that experiments are the most common form of study validation aimed at identifying the effects of TDD on internal software quality.

When experiments were used as a validation method, there is a clear difference in findings between the academic and the industrial settings. The experiments that were applied in the industrial setting, it was found that TDD increased internal software quality, while in the academic setting, most studies (about 60%) also found that TDD helps increase internal software quality, but the studies by [31] and [42] did not identify any differences between the use of the two practices, while the studies by [28] and [35] pointed out that the TDD practice decreased internal quality compared with TLD.

When the works by [26] and [40] were considered as single studies, there was a total of 21 studies that have made the comparison between TDD and TLD. About 76% of this total reported a significant improvement in internal software quality. When the repetitions are taken into account, this total rises to 80%.

4.3. External quality

Table 6 shows the list of studies and the effects on external software quality. The main indicator used for identifying the effects was black box testing, in order to measure the amount of test cases that passed or failed in accordance with the practice in use (TDD and TLD).

In Table 6, the studies contained in the column **(+) EQ** are those which tested positive, and there was an increase in external software quality when the TDD practice was applied. Column **(=) EQ** shows the studies that have not identified any effect when the two practices were applied, while column **(−) EQ** contains those which found negative effects on external quality by applying the TDD practice, i.e., external quality decreased.

When Table 6 is analyzed in detail, it shows that the experiments and case studies are the most common forms of study validation for identification of the effects of TDD on external software quality.

The study by [23] identified that external software quality remained the same when comparing the TDD with the TLD practices. The study by [28] found that when the TDD practice was used, external software quality decreased. For all other studies listed in Table 6, there was evidence that TDD helped improve external software quality.

Considering only single papers, that is, counting the study by [40] only once, 16 studies made a comparison between TDD and TLD. About 88% of this total reported a significant improvement in external software quality.

Table 7
Studies that show the effects of TDD on productivity.

Method	Setting	(+) Prd.	(=) Prd.	(−) Prd.
Experiment	Academic	[10,23,35]	[21,31,36]	[27]
Experiment	Industrial		[25]	[22,24,32]
Case Study	Academic	[43,45]		
Case Study	Industrial		[20]	[33,37,44]
Simulation	Industrial			[34]

4.4. Productivity

Table 7 lists the studies and the effects on productivity. The main indicator used to determine if there was a change in productivity was to compare the time taken to implement a new functionality and the amount of generated code.

In Table 7, the papers contained in the column **(+) Prd.** are the ones that showed an increase in developer productivity when the TDD practice was applied. The column **(=) Prd.** shows the papers that have not identified any effect when applying the two practices. By contrast, the column **(−) Prd.** contains those that reported a decrease in developer productivity when the TDD practice was applied.

The results shown in Table 7 are contradictory when comparing the differences between the industrial and the academic settings. None of the studies applied in an industrial setting showed an increase in productivity; it remained constant at most, while for the studies that were applied in the academic setting, only [27] showed a decrease in productivity.

Only the studies by [10,23,35,43] and [45], applied in an academic setting, pointed out that TDD increased productivity in software development. Productivity was the item that generated the most outstanding results, but most studies pointed to a decrease in productivity when applying TDD, compared with the TLD practice.

As shown in Table 7, 18 studies made the comparison between TDD and TLD by taking productivity into consideration. About 44% of this total reported that developer productivity decreased; the other studies indicated that productivity either remained stable or increased (28% each).

5. Threats to validity

In this section, the threats to the validity of the present review are discussed, according to the five general validity ratings (theoretical validity, generalizability, objectivity, interpretative validity and repeatability) defined in [47].

Theoretical validity: Theoretical validity aims to identify the possible presence of factors that may lead to confusion in the analysis. Publication bias, whether towards positive or negative results, can have effects on the overall outcome of the analysis. This may go unnoticed by researchers.

One way to reduce or even eliminate this bias is to increase the number of samples that make up the literature review. For this review, 27 papers were selected and the results follow a similar direction, hence this bias does not appear to be a threat to this study.

The studies using the Simulation and Questionnaire as research methods have a small sample (three studies in total). In this case, an isolated assessment of these three studies could pose a threat to theoretical validity, but in this study, the comparison was performed considering all the works.

Generalization: It is the ability to expand and generalize the results to different environments, settings and situations. In this aspect, the number of experiments and case studies is high, and the application was made in various contexts (academic and industrial).

This threat is low when considering all the studies discussed in this review. However, the application of research in the industry using real-world projects was performed at a small scale, and further research is necessary to expand the results found so far.

Objectivity: It is the ability to correctly and faithfully describe what has been observed as regards the identified aspects. To reduce the risk of including or excluding studies that are relevant to this review, the criteria for inclusion and exclusion were defined in Section 3.4 by clearly stating how the works were collected and excluded from the final list. This process was performed twice by the same researcher, but there has been no inter-researcher validation.

The fact that the process of selecting and extracting information from the papers was performed by a single investigator is considered to be a low threat. However, this process was implemented and revised twice in order to lower the risk of misinterpretation of results, but this threat cannot be completely removed.

Interpretative validity: It is focused on making the right conclusions from the included data in an objective manner. This study analyzed the data and the information reported in all studies, extracting only the results and reports that had been validated in the studies. The lack of standardization in the metrics used in the studies made it difficult to extract information during the preparation of this review. This threat is considered to be low since the information was revised twice.

Repeatability: It focuses on a clear definition of attributes, instruments and actions to assist in the repetition of the study. Thus, another researcher that follows the review of the protocol is expected to access the studies and reach similar conclusions.

To reduce this threat, the entire process, protocol and research method were described in detail in Section 3, allowing this work to be replicated and extended in the future by other researchers. The process of systematic review in the present paper followed the guidelines presented in [12–14] e [16].

6. Discussion

Studies conducted in order to compare TDD with TLD showed a clear division into two groups as for settings; namely, academic and industrial. As shown in Table 3, the share of studies in this category is balanced: 48.14% in the academic setting and 44.46% in the industrial setting.

Regardless of the setting where the study was carried out, Java technology is the most widely programming language used in the studies. One of the main reasons is the popularity of the language and the number of tools that support the TDD practice. The research methods most often used in the studies were the experiment and the case study.

One factor that made it difficult to completely analyze and compare the results is that each study used different metrics and indicators to extract information. In this case, this review considered the effects identified in the analyzed studies regardless of the metrics used.

Most studies indicate an improvement in internal software quality; only the studies by [28] and [35] showed a decrease in internal quality.

However, it should be noted that source code coverage was the metric most commonly used as an indicator to determine internal quality. Code coverage determines the amount of unit tests to cover the productive source code of the system.

A 100% coverage determines that the production code is completely covered by tests, but it does not guarantee that the unit tests are properly implemented or that the system will have no defects.

Therefore, other metrics should be analyzed by these studies to determine if internal software quality is actually increasing. Five out of the 27 studies used other metrics relative to internal quality, usually applied in the development community, such as code complexity metrics (McCabe), Lack of Cohesion in Methods (LCOM), Single Responsibility Principle (SRP), Open-Closed Principle (OCP) and Coupling Between Objects (CBO). Studies that observed these other metrics are [20,26,35,38] and [10]. Among the five reviewed studies, only [20] and [26] used real-world projects for applying the case study.

Regarding external quality, only the study by [28] pointed out that there was a decrease in quality; the other studies showed an increase or at least constant external software quality. In this item, the form of measurement was almost the same in all the works, and it was based on the number of test cases that failed or passed at the end of the software development process, generating a defect density for each practice that was executed. In the analyses, the TDD practice promoted an increase in external quality.

The analysis of the results on productivity showed a clear a difference difference across the collected data, depending on the setting where the study was applied. Only one out of the nine studies in the academic setting, the study by [27] pointed to a decrease in productivity. The other studies found that productivity remained stable or increased.

By contrast, in the industrial setting, only the studies by [25] and [20], out of the nine studies, indicated that productivity remained the same; the others pointed to a decrease in productivity. Productivity was measured considering the time taken for a feature to be implemented using the TDD and the TLD techniques.

The study [24] pointed out that there is a moderate correlation between working time and resulting quality. Perhaps this correlation can justify the decrease in productivity.

None of the studies considered conducting the analyses and comparisons in legacy projects, but rather in new projects only.

7. Related works

During the searches on the TDD practice in software development, five literature reviews were found. They bring significant contributions to this field of research. These studies were analyzed, and this section presents an overview of each study.

The study by Causevic et al. [13] discusses seven factors that limit the use of TDD in the industry. These factors are: (1) increase in development time; (2) employees' lack of knowledge of TDD; (3) insufficient details in the design project; (4) developers' insufficient knowledge of testing; (5) lack of adherence to the TDD protocol; (6) limitation of tools and specific domains; (7) legacy code.

The review by Jeffries and Melnik [48] listed studies on the impact of TDD on productivity and quality of work in the industrial and the academic settings until 2007. The studies showed controversial results both in the academy and in the industry. However, the authors mentioned the difficulty in comparing the findings from the articles they analyzed because each study used different metrics.

The study by Rafique and Misic [5] presents a meta-analysis in order to investigate the impact of TDD on external quality and productivity in software development. By using statistical techniques to combine the results of various studies into a single measure, the authors observed a variation in the application of the TDD practice in academic settings and in the industry.

A systematic review by Sfetsos and Stamelos [49] sought to address other techniques from the agile methods in addition to TDD. Among the 46 studies they analyzed, 18 of them were on TDD, distributed as follows: eight reports of experiments, eight case studies, and two studies with mixed approaches. These studies showed that the TDD practice improved external software quality by reducing the number of defects.

Table 8
List of systematic review of studies on TDD.

Ref.	Year	N° Studies	Results
[13]	2011	48	Paper showed 18 effects that TDD can generate, and seven of them were classified as limitations for adoption of TDD in the industry.
[48]	2007	18	Despite the conflicting results, most studies show an increase in quality and a decrease in productivity.
[5]	2013	27	There was increased quality in academic studies in the industry but the result was not conclusive. Productivity had a higher decrease in the industrial setting than in the academic one.
[49]	2010	46	Increase in external software quality, but the results on productivity and internal quality were contradictory.
[11]	2014	41	Clear results of increased external quality along with a slight decrease in productivity.

In the review by Munir et al. [11], the authors classified the studies according to the levels of rigor and relevance of each paper. In the same study, eight classification variables were defined and in accordance with the findings, there is good evidence of increased external software quality, but this does not directly result in decreased costs.

Table 8 shows an overview of the literature related to the present review. The column **Ref.** shows the references of the analyzed article, the column **Year** indicates the year of publication and the column **N°Studies** shows the number of studies analyzed in this paper. Finally, the column **Results** summarizes the main findings and results presented by the paper in question.

Most of these studies focused on looking at the effects that the TDD practice produced on external quality and productivity in software development. The present paper is particularly focused on the effects that TDD can generate in productivity and internal and external quality, bringing up a concern from the perspective of software architects and developers.

8. Conclusion and further research

This systematic review was conducted with the aim of locating, selecting and evaluating available empirical studies on the application of TDD in software development, targeting the effects that this practice produces on productivity and internal and external software quality.

Therefore, the studies are mapped in Table 3, and the impacts are analyzed in Section 4, while Sections 4 and 6 present the details of the effects produced by the TDD practice on software development and also the comparison and classification of the studies included in this systematic review.

The analysis of the state of the art of this review suggests that there is a demand for research using the TDD practice in technologies and development paradigms other than object orientation.

The sources of information were electronic knowledge bases, academic papers and conference proceedings available online. When the studies classified in the tables in Section 4 are considered together, most studies point to an increase in internal and external software quality. As regards productivity, there is a difference: while in the academic setting the results point to an increase in productivity, they signal a decrease in the industrial setting.

This analysis also reveals the need for case studies or experiments to be applied in a real software development setting while using real-world projects, rather than fictitious projects only.

However, as reported by [21], evaluating the effects of TDD proved to be challenging in the industrial setting, mainly because of the difficulty in finding industry professionals willing to volunteer for the experiments.

Finally, the present review identified six opportunities for further research that would have great importance in this field:

- Scientifically validate the hypothesis presented by authors that TDD improves the quality of the written source code and productivity;
- Extend the TDD practice to other paradigms and software development technologies, as most research is focused on simple examples using the Java language;
- Conduct further research using real projects in corporate environments for more realistic results;
- Investigate the adoption of TDD and its effects on legacy systems, as most of the studies are focused on new systems;
- Produce tools that simplify the adoption of TDD on new development paradigms;
- Use a set of code analysis metrics to validate internal quality; the most commonly used metric is code coverage.

The aforementioned opportunities show that there are still some gaps that need to be filled by the software engineering research community to clarify and provide new evidence about the effects produced by the TDD practice on software design.

References

[1] VersionOne, 8th Annual State of Agile Survey, Technical Report, VersionOne Inc., 2014.

[2] C.d.O. Melo, V. Santos, E. Katayama, H. Corbucci, R. Prikladnicki, A. Goldman, F. Kon, The evolution of agile software development in Brazil, J. Braz. Comput. Soc. 19 (4) (2013) 523–552, doi:10.1007/s13173-013-0114-x.

[3] K. Beck, Test Driven Development: By Example, first ed., Addison-Wesley, Boston, 2002.

[4] K. Beck, Extreme Programming Explained: Embrace Change, first ed., Addison-Wesley, Boston, 1999.

[5] Y. Rafique, V.B. Misic, The effects of test-driven development on external quality and productivity: A meta-analysis, IEEE Trans. Softw. Eng. 39 (6) (2013) 835–856, doi:10.1109/TSE.2012.28.

[6] B. Koltai, J. Warnick, R. Agbaji, S. Nilan, Test-driven development, ACM Trans. Comput. Logic (TOCL) V (2011) 1–21.

[7] V. Massol, P. Tahchiev, F. Leme, G. Gregory, Component-Based Software Engineering: Putting the Pieces Together, second ed., Manning Publications, Stamford, 2010.

[8] K. Beck, Aim, fire [test-first coding], IEEE Softw. 18 (5) (2001) 87–89, doi:10.1109/52.951502.

[9] L.-O. Damm, L. Lundberg, D. Olsson, Introducing test automation and test-driven development: An experience report, Electron. Notes Theor. Comput. Sci. 116 (2005) 3–15, doi:10.1016/j.entcs.2004.02.090.

[10] M. Pancur, M. Ciglaric, Impact of test-driven development on productivity, code and tests: A controlled experiment, Inf. Softw. Technol. 53 (6) (2011) 557–573 Special Section: Best papers from the {APSEC} Best papers from the {APSEC}, doi:10.1016/j.infsof.2011.02.002.

[11] H. Munir, M. Moayyed, K. Petersen, Considering rigor and relevance when evaluating test driven development: A systematic review, Inf. Soft. Technol. 56 (4) (2014) 375–394, doi:10.1016/j.infsof.2014.01.002.

[12] B. Kitchenham, O.P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering : A systematic literature review, Inf. Softw. Technol. 51 (1) (2009) 7–15 Special Section - Most Cited Articles in 2002 and Regular Research Papers, doi:10.1016/j.infsof.2008.09.009.

[13] A. Causevic, D. Sundmark, S. Punnekkat, Factors limiting industrial adoption of test driven development: A systematic review, in: Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST), 2011, pp. 337–346, doi:10.1109/ICST.2011.19.

[14] B. Kitchenham, S. Charters, Guidelines for performing Systematic Literature Reviews in Software Engineering, Technical Report, EBSE 2007-001, Keele University and Durham University, 2007.Joint Report

[15] B. Kitchenham, Procedures for Performing Systematic Reviews, Technical Report, Departament of Computer Science, Keele University, 2004.

[16] M. Petticrew, H. Roberts, Systematic Reviews in the Social Sciences: A Practical Guide, first ed., Wiley-Blackwell, Oxford, 2005.

[17] A. Tahir, D. Tosi, S. Morasca, A systematic review on the functional testing of semantic web services, J. Syst. Softw. 86 (11) (2013) 2877–2889, doi:10.1016/j.jss.2013.06.064.

[18] S. Jalali, C. Wohlin, Systematic literature studies: Database searches vs. backward snowballing, in: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '12, ACM, New York, NY, USA, 2012, pp. 29–38, doi:10.1145/2372251.2372257.

[19] T. Dyba, T. Dingsoyr, G.K. Hanssen, Applying systematic reviews to diverse study types: An experience report, in: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, ESEM '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 225–234, doi:10.1109/ESEM.2007.21.

[20] M. Siniaalto, P. Abrahamsson, A comparative case study on the impact of test-driven development on program design and test coverage, in: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, ESEM '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 275–284, doi:10.1109/ESEM.2007.2.

[21] A. Geras, M. Smith, J. Miller, A prototype empirical evaluation of test driven development, in: Proceedings 10th International Symposium on Software Metrics, 2004., 2004, pp. 405–416, doi:10.1109/METRIC.2004.1357925.

[22] B. George, L. Williams, A structured experiment of test-driven development, Inf. Softw. Technol. 46 (5) (2004) 337–342 Special Issue on Software Engineering, Applications, Practices and Tools from the {ACM} Symposium on Applied Computing 2003, doi:10.1016/j.infsof.2003.09.011.

[23] A. Gupta, P. Jalote, An experimental evaluation of the effectiveness and efficiency of the test driven development, in: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, ESEM '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 285–294, doi:10.1109/ESEM.2007.20.

[24] B. George, L. Williams, An initial investigation of test driven development in industry, in: Proceedings of the 2003 ACM Symposium on Applied Computing, SAC '03, ACM, New York, NY, USA, 2003, pp. 1135–1139, doi:10.1145/952532.952753.

[25] E.M. Maximilien, L. Williams, Assessing test-driven development at IBM, in: Proceedings of the 25th International Conference on Software Engineering, ICSE '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 564–569, doi:10.1109/ICSE.2003.1201238.

[26] D. Janzen, H. Saiedian, Does test-driven development really improve software design quality? IEEE Softw. Mag. 25 (2) (2008) 77–84, doi:10.1109/MS.2008.34.

[27] J.H. Vu, N. Frojd, C. Shenkel-Therolf, D.S. Janzen, Evaluating test-driven development in an industry-sponsored capstone project, in: Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations, ITNG '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 229–234, doi:10.1109/ITNG.2009.11.

[28] M. Pancur, M. Ciglaric, M. Trampus, T. Vidmar, Towards empirical evaluation of test-driven development in a university environment, in: Proceedings of the IEEE Region 8. EUROCON 2003. Computer as a Tool, vol. 2, 2003, pp. 83–86, doi:10.1109/EURCON.2003.1248153.

[29] D.S. Janzen, C.S. Turner, H. Saiedian, Empirical software engineering in industry short courses, in: Proceedings of the 20th Conference on Software Engineering Education Training, CSEET '07., 2007, pp. 89–96, doi:10.1109/CSEET.2007.20.

[30] S.H. Edwards, Using software testing to move students from trial-and-error to reflection-in-action, in: Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '04, ACM, New York, NY, USA, 2004, pp. 26–30, doi:10.1145/971300.971312.

[31] D. Janzen, H. Saiedian, Test-driven learning in early programming courses, in: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '08, ACM, New York, NY, USA, 2008, pp. 532–536, doi:10.1145/1352135.1352315.

[32] G. Canfora, A. Cimitile, F. Garcia, M. Piattini, C.A. Visaggio, Evaluating advantages of test driven development: A controlled experiment with professionals, in: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering, ISESE '06, ACM, New York, NY, USA, 2006, pp. 364–371, doi:10.1145/1159733.1159788.

[33] T. Bhat, N. Nagappan, Evaluating the efficacy of test-driven development: Industrial case studies, in: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering, ISESE '06, ACM, New York, NY, USA, 2006, pp. 356–363, doi:10.1145/1159733.1159787.

[34] I. Turnu, M. Melis, A. Cau, A. Setzu, G. Concas, K. Mannaro, Modeling and simulation of open source development using an agile practice, J. Syst. Archit. 52 (11) (2006) 610–618 Agile Methodologies for Software Production, doi:10.1016/j.sysarc.2006.06.005.

[35] D.S. Janzen, H. Saiedian, On the influence of test-driven development on software design, in: Proceedings of the 19th Conference on Software Engineering Education & Training, CSEET '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 141–148, doi:10.1109/CSEET.2006.25.

[36] H. Erdogmus, M. Morisio, M. Torchiano, On the effectiveness of the test-first approach to programming, IEEE Trans. Serv. Comput. 31 (3) (2005) 226–237, doi:10.1109/TSE.2005.37.

[37] J.C. Sanchez, L. Williams, E.M. Maximilien, On the sustained use of a Test-Driven Development practice at IBM, in: Proceedings of the Agile Conference (AGILE), 2007, pp. 5–14, doi:10.1109/AGILE.2007.43.

[38] M.F. Aniche, M.A. Gerosa, How the practice of TDD influences class design in object-oriented systems: Patterns of unit tests feedback, in: Proceedings of the 26th Brazilian Symposium on Software Engineering, SBES '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 1–10, doi:10.1109/SBES.2012.14.

[39] O.P.N. Slyngstad, J. Li, R. Conradi, H. Rønneberg, E. Landre, H. Wesenberg, The impact of test driven development on the evolution of a reusable framework of components – an industrial case study, in: Proceedings of the 2008 Third International Conference on Software Engineering Advances, ICSEA '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 214–223, doi:10.1109/ICSEA.2008.8.

[40] M.F. Aniche, M.A. Gerosa, Most common mistakes in test-driven development practice: Results from an online survey with developers, in: Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, ICSTW '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 469–478, doi:10.1109/ICSTW.2010.16.

[41] L.-O. Damm, L. Lundberg, Results from introducing component-level test automation and test-driven development, J. Syst. Softw. 79 (7) (2006) 1001–1014 Selected papers from the 11th Asia Pacific Software Engineering Conference (APSEC2004), doi:10.1016/j.jss.2005.10.015.

[42] L. Madeyski, The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment, Inf. Softw. Technol. 52 (2) (2010) 169–184, doi:10.1016/j.infsof.2009.08.007.

[43] L. Madeyski, L. Szala, The impact of test-driven development on software development productivity – an empirical study, in: P. Abrahamsson, N. Baddoo, T. Margaria, R. Messnarz (Eds.), Software Process Improvement, *Lecture Notes in Computer Science*, vol. 4764, Springer Berlin Heidelberg, 2007, pp. 200–211, doi:10.1007/978-3-540-75381-0_18.

[44] N. Nagappan, E.M. Maximilien, T. Bhat, L. Williams, Realizing quality improvement through test driven development: Results and experiences of four industrial teams, Empir. Softw. Eng. 13 (3) (2008) 289–302, doi:10.1007/s10664-008-9062-z.

[45] S. Yenduri, L.A. Perkins, Impact of using test-driven development: A case study, in: H.R. Arabnia, H. Reza (Eds.), Proceedings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP, vol. 1, CSREA Press, Las Vegas, Nevada, USA, 2006, pp. 126–129.

[46] L. Madeyski, L. Szala, The impact of test-driven development on software development productivity – an empirical study, in: P. Abrahamsson, N. Baddoo, T. Margaria, R. Messnarz (Eds.), Software Process Improvement, *Lecture Notes in Computer Science*, vol. 4764, Springer Berlin Heidelberg, 2007, pp. 200–211, doi:10.1007/978-3-540-75381-0_18.

[47] K. Petersen, C. Gencel, Worldviews, research methods, and their relationship to validity in empirical software engineering research, in: Proceedings of the Joint Conference of the 23rd International Workshop on Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013, pp. 81–89, doi:10.1109/IWSM-Mensura.2013.22.

[48] R. Jeffries, G. Melnik, TDD - the art of fearless programming, IEEE Softw. 24 (3) (2007) 24–30, doi:10.1109/MS.2007.75.

[49] P. Sfetsos, I. Stamelos, Empirical studies on quality in agile practices: A systematic literature review, in: Proceedings of the Seventh International Conference on the Quality of Information and Communications Technology (QUATIC), 2010, pp. 44–53, doi:10.1109/QUATIC.2010.17.