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.