An External Replication on the Effects of Test-driven Development Using a Multi-site Blind Analysis Approach

Davide Fucci
M-Group, University of Oulu
Oulu, Finland
davide.fucci@oulu.fi

Giuseppe Scanniello
University of Basilicata
Potenza, Italy
giuseppe.scanniello@unibas.it

Simone Romano
University of Basilicata
Potenza, Italy
simone.romano@unibas.it

Martin Shepperd
Brunel University
London, United Kingdom
martin.shepperd@brunel.ac.uk

Boyce Sigweni
Brunel University
London, United Kingdom
boyce.sigweni@brunel.ac.uk

Fernando Uyaguari
Universidad Politécnica de Madrid
Madrid, Spain
f.uyaguari@alumnos.upm.com

Burak Turhan
M-Group, University of Oulu
Oulu, Finland
burak.turhan@oulu.fi

Natalia Juristo
M-Group, University of Oulu
Universidad Politécnica de Madrid
natalia.juristo@oulu.fi

Markku Oivo
M-Group, University of Oulu
Oulu, Finland
markku.oivo@oulu.fi

ABSTRACT

Context: Test-driven development (TDD) is an agile practice claimed to improve the quality of a software product, as well as the productivity of its developers. A previous study (i.e., baseline experiment) at the University of Oulu (Finland) compared TDD to a test-last development (TLD) approach through a randomized controlled trial. The results failed to support the claims. *Goal:* We want to validate the original study results by replicating it at the University of Basilicata (Italy), using a different design. *Method:* We replicated the baseline experiment, using a crossover design, with 21 graduate students. We kept the settings and context as close as possible to the baseline experiment. In order to limit researchers bias, we involved two other sites (UPM, Spain, and Brunel, UK) to conduct blind analysis of the data. *Results:* The Kruskal-Wallis tests did not show any significant difference between TDD and TLD in terms of testing effort (*p-value = .27*), external code quality (*p-value = .82*), and developers' productivity (*p-value = .83*). Nevertheless, our data revealed a difference based on the order in which TDD and TLD were applied, though no carry over effect. *Conclusions:* We verify the baseline study results, yet our results raises concerns regarding the selection of experimental objects, particularly with respect to their interaction with the order in which of treatments are applied. We recommend future studies to survey the tasks used in experiments evaluating TDD. Finally, to lower the cost of replication studies and reduce researchers' bias, we encourage other research groups to adopt similar multi-site blind analysis approach described in this paper.

ESEM '16, September 08 - 09, 2016, Ciudad Real, Spain

ACM 978-1-4503-4427-2/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2961111.2962592

Keywords

test-driven development, external experiment replication, blind analysis

1. INTRODUCTION

Test-driven development (TDD) is an agile practice [4], in which unit tests are written before production code, compelling the developer to focus on the correct behavior of the intend feature from an early stage. The developer writes just enough production code to make the unit test pass, and refactors it afterwards [4].

A TDD cycle is composed of three, iterative phases.
Red: Write a unit test; the unit test fails.
Green: Write production code; the unit test passes.
Blue: Refactor the code [11]; the unit test passes.
The SE research community has taken an interest in investigating the effects of TDD on several outcomes, including testing effort, external software quality and developers' productivity. In particular, there is a plethora of controlled experiments (e.g., [10, 18, 27, 28]), aggregated in secondary studies (e.g., [32, 36, 39]), which report mostly inconclusive, or inconsistent results. A number of experiments were carried out at the University of Oulu [13, 16]—including replication studies [14, 15]—investigating the same outcomes. In order to understand the reliability of the findings in [13] (i.e., *baseline experiment*), we carried out a close, external replication [24] at the University of Basilicata, Potenza (Italy). Authors GS and SR conducted the experiment us-

ing the protocol devised by the original experimenters (i.e., authors DF, BT and MO), but with a different design—i.e., crossover. In order to limit the possible threat to validity due to researchers' bias [26], the data analysis was done in different sites. The data was extracted at one site (i.e., UPM, Madrid) and analyzed at another site (i.e., Brunel University, London). Both sites' researchers were not aware of the experiment goal. Through this replication we checked the validity of the results presented in the baseline experiment, and expanded them by analyzing the impact of the changes made to the experiment design and settings.

The paper is organized as follows: in Section 2, we present a summary of the previous studies regarding TDD, and the salient literature about external replication and blind analysis. In Section 3, we describe the baseline experiment. In Section 4, we present the goals, changes in context, data analysis strategy, and results of the replication. The threats to validity are discussed in Section 5. In Section 6, we discuss results and compare them to those of the baseline experiment. Final remarks and future work conclude the paper.

2. BACKGROUND

In this section, we report the results of the secondary studies investigating the effects of TDD on developers' productivity and external quality. In order to position our work, we report a brief review about external replication and blind analysis for software engineering.

2.1 Previous studies on TDD

The systematic review reported by Turhan *et al.* [41] includes 32 primary studies, published between 2000 and 2008, in which TDD is compared to different, more traditional development approaches. The primary studies vary in type (i.e., controlled experiment, pilot studies, and industrial use), and were classified according to the experience of the participants, the level of details reported for the constructs, and the scale of the study. When taking into account all the studies, the authors suggest that TDD improves external quality. However, this result does not hold once only the rigorous primary studies are taken into account. The results regarding productivity vary across the type of primary studies, from positive, to negative, to inconclusive. Nevertheless, the controlled experiments show improvements due to the use of TDD. The meta-analysis by Rafique and Mišić [36] includes 27 primary studies published until 2011. The results, based on effects sizes [12], do not show solid evidence either in support, or against, TDD. Interestingly, when considering the academic context (i.e., the same as the baseline study, and this replication), TDD improves external quality of 24% with respect to a waterfall approach;[1] however, its effects are detrimental (-13%) when compared to an iterative test-last (ITL) approach (i.e., a similar approach is used by the control groups in the baseline study and this replication). Conversely, TDD increases productivity—27% vis-à-vis ITL, and 7% vis-à-vis waterfall development. The authors recommend the use of an ITL-based approach to be compared against TDD in order to remove the extra testing effort required by, for example, a waterfall approach. Munir *et al.*'s systematic review [32] covered the 2000-2011 decade, and included 41 primary studies categorized by rigor and relevance according to the framework presented in [22]. The results are inconclusive when the primary studies are considered together; however the nine high-rigor, high-relevance studies show that TDD improves external quality, while productivity is not affected. The 21 studies in the same category of the baseline experiment and this replication (i.e., low-relevance due to its academic context, but high-rigor due level of control) are inconclusive with respect to both outcomes. None of the secondary studies survey the effects of TDD on testing effort.

[1]Using waterfall, the system is tested once it is considered completely developed.

2.2 External replications in SE

The software engineering community has been readily embracing replications (e.g., [9, 25, 38]). There are two important factors characterizing a replication: *procedure* (i.e., followed experimental steps) and *researcher* (i.e., who conducted the replication). As for the procedure, the kinds of replications range in between *conceptual* and *close* [38]. A replication is conceptual if the research questions are the same as the baseline experiment, but the experimental procedures are completely different. A replication is close if the baseline experiment and the replication share the same procedure [38]. As for the researcher, we distinguish between *internal* and *external*. An internal replication is conducted by the same group of researchers as the baseline experiment [30], while an external replication is performed by different experimenters. da Silva *et al.* systematic review [9] showed that the majority of replications between 1994 and 2010 were internal. Alongside, internal replications tend to report positive results not only in SE [7], but also in other disciplines [1]. Therefore, external replications should be preferable as they are less affected by experimenters' biases [24, 38]. More recently, Gómez *et al.* [19] propose a different classification, in which the study presented in this paper occurs to be an *operational* replication. In particular, following Gómez *et al.*, we consider this to be a changed-experimenter replication, because at least two treatments and one response variable are retained, and the researchers are different with respect to the baseline study. Independently from the classification for types of replications, there are two primary motivations to perform replications: *(i)* they are necessary to solve problems and to collect evidence because they bring credibility to a given research and *(ii)* they are valuable because they provide evidence on the benefits of a software engineering practice thus allowing industrial stakeholders to use this information to support adoption decisions [2, 19, 35].

2.3 Blind analysis

Another strategy to minimize the possibility of bias in experimental results is blind analysis [40]. Clearly for blind analysis to be effective it requires a minimum of two researchers, one who carries out the experiment (or simply experimenter) and another who conducts data analysis (or simply analyst). The main idea is that, by relabeling (or anonymizing) the different treatments, the analyst is no longer aware of which is the new (or known experimented) technique/approach, nor which are the results from benchmarks. Searching for a test or procedure that yields statistical significance is less straightforward. It is more difficult for the analyst to have a view as to what results are "interesting." This kind of analysis is almost unknown in software engineering (e.g., [37, 40], unlike other disciplines (e.g., [3]).

3. BASELINE EXPERIMENT (UOULU)

We provide the context for the replication by reporting the key information about the baseline experiment. We followed the recommendations in [6]. The baseline study [13] itself reports a replication of a previous study by Erdogmus *et al.* [10][2]. Both studies were structured in two parts. The first part covered a controlled experiment to assess the impact of TDD[3] on testing effort, productivity and external quality. The experiment compared TDD to a control composed of developers applying test-last development (TLD) method, while working on a single task. The TLD developers followed an approach in which unit tests are written only after some production code (e.g., the code necessary for a feature of the task) was present. They were asked to work on small chunks of the task, rather than utilize a *waterfall* approach; accordingly the task was designed to support this way of working. The second part reported a correlational study investigating the relationship between testing effort and the other two outcomes.[4] In this paper, we report a replication of the first phase of Fucci *et al.*'s study [13].

3.1 Research Questions and Hypotheses

The research questions driving the part of the baseline study replicated in this paper were:

Do test-first developers write more tests than test-last developers?

Do test-first developers produce solutions with higher external quality than test-last developers?

Are test-first developers more productive than test-last developers?

The independent variable (IV) was, therefore, development approach (i.e., TDD or TLD), whereas the dependent variables (DV) were: testing effort ($TESTS$), software external quality ($QLTY$), and developers' productivity ($PROD$). Note that, following from the research questions, the hypotheses were formulated as *directional* for consistency with Erdogmus *et al.*, but analyzed as non-directional since the existing body of knowledge regarding the postulated impact of TDD does not suggest a specific direction of the effect. The UOULU's three causal research hypotheses were formulated according to the research questions. Table 1 reports the original hypotheses, as well as their outcomes. [13]. Some of the participants in the original study used pair-programming (PP) along with TDD or TLD. Therefore, the original study included an additional hypothesis dealing with the possible interactions between the practices under study, and PP as nuisance factor. In our replication, participants worked individually; therefore such additional hypothesis was not taken into account.

3.2 Sample and participants

The sample in the original study was composed of 58 participants (33 graduate and 25 undergraduate), enrolled at the department of Information Processing Science, University of Oulu, and attending the Fall 2012 course *Software Quality and Testing*. The participants were sampled by convenience. Before the experiment, a pre-questionnaire was administered to the participants in order to grasp their experience. Fifteen participants reported to have professional experience varying between three months and ten years, with an average of 2.6 years. All the participants, before the study, received 20 hours of training in unit testing in Java, using JUnit as a framework and Eclipse as the IDE. The training was completely practice oriented and required the participants to solve a variety of programming katas[5] using TDD. The participants were informed about their participation to the experiment; however, the research questions were not disclosed to them in advance. Some of the participants used pair-programming: 11 pairs were formed; the remaining 36 participants worked individually. In the final design seven pairs and 20 individuals were allocated to the TDD group; four pairs and 16 individuals were allocated to the TLD group. Hence, the design was unbalanced.

3.3 Object

The baseline experiment utilized a *single* experimental object: the Bowling Scorekeeper (BSK) kata. The task required to participants to implement an API for calculating the score of a player in a bowling game. The development of a GUI was not required. The task was divided into 13 user stories of incremental difficulty, each building on the results of the previous one. An example, in terms of input and expected output, accompanied the description of each user story. An Eclipse project, containing a stub of the expected API signature (51 Java SLOC); also an example JUnit test (9 Java SLOC) was provided together with the task description. The task was administered electronically at the beginning of the experiment sessions.

3.4 Metrics

The relevant metrics for this study are $TEST$, $QLTY$, and $PROD$. $TEST$ is defined as the number of JUnit assert statements within the unit test suite written by the participants while tackling the task. Therefore, $TEST \in [0, \infty]$. $QLTY$ represents how well, on average, the implementation of each user story matches that user story's requirements according to a pre-defined set of acceptance tests developed by the authors. $PROD$ represents the portion of the task correctly implemented. Both $QLTY$ and $PROD$ were calculated based on an acceptance test suite developed by the researchers, and hidden to the participants. In particular, $QLTY$ is computed as follows:

$$QLTY = \frac{\sum_{i=1}^{\#TUS} QLTY_i}{\#TUS} \times 100, \quad (1)$$

where $\#TUS$ represents how many user stories a participant tackled. A user story is considered tackled if at least one assert in any of the acceptance tests associated with that user story passes. Hence, $\#TUS$ is formally defined as:

$$\#TUS = \sum_{i=0}^{n} \begin{cases} 1 & \#ASSERT_i(PASS) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

[2]Acknowledging Erdogmus *et al.* [10] as the first study in our family of experiments, we use the terms *original study* or *baseline study* interchangebly based on context for referring to Fucci *et al.* [13] within the scope of this paper.

[3]The baseline experiment refers to it as test-first for consistency with the Erdogmus *et al.*

[4]The rationale for it is based on the conjecture that TDD developers produce more unit tests, hence a positive correlation between $TESTS$ and $QLTY$ or $PROD$ was expected.

[5]A code kata is a programming exercise used to practice a technique or language.

Table 1: UOULU hypotheses and outcomes (from [13])

Name	H_0	H_1	Outcome H_0
1T	$TEST(TDD) = TEST(TLD)$	$TEST(TDD) > TEST(TLD)$	Failed to reject
1Q	$QLTY(TDD) = QLTY(TLD)$	$QLTY(TDD) > QLTY(TLD)$	Failed to reject
1P	$PROD(TDD) = PROD(TLD)$	$PROD(TDD) > PROD(TLD)$	Failed to reject

For the BSK task, $n = 13$. The quality of the *i-th* user story, $QLTY_i$, is defined as:

$$QLTY_i = \frac{\#ASSERT_i(PASS)}{\#ASSERT_i(ALL)}. \quad (3)$$

From Formulas 1-3, $QLTY \in [0, 100]$. The metric for $PROD$ was calculated as follows:

$$PROD = \frac{\#ASSERT(PASS)}{\#ASSERT(ALL)} \times 100, \quad (4)$$

in other words $PROD$ represents the percentage of assert statements in the acceptance test suite passed. From Formula 4, $PROD \in [0, 100]$.

3.5 Design, Context and Execution

The baseline experiment was designed as one factor (i.e., development approach), with two treatments (i.e., TDD and TLD) [45]. No blocking was applied; nevertheless the effect of a nuisance factor, pair-programming, was assessed after-the-fact. At the beginning of the experiment session, the participants were randomly assigned to either groups (TDD or TLD). The sample was not stratified before the assignment to the groups. The researchers distributed the material (i.e., hard and soft copies of the experimental task, and Eclipse template) to each group accordingly. The only difference between the two groups was whether the task required to be implemented using TDD or TLD. The session lasted for three hours, after which the participants returned the Eclipse project used to develop their solutions. These artifacts were later used to extract the metrics.

3.6 Summary of results

The results of the original study are reported in Table 1. Mann-Whitney U-test [21] was used to test hypotheses 1T, 1Q and 1P. The null-hypothesis was failed to be rejected in each case. The effect size, reported as Pearson's r [34], are small; respectively .16, .04 and .11. The conclusion of the original study is that TDD does not affect testing effort, external software quality, and developers' productivity when compared to TLD.

4. EXTERNAL REPLICATION (UNIBAS)

4.1 Motivations

The replication was carried out following the guidelines reported in [23]. We wanted to validate the results of the UOULU regarding TDD in an academic context; in order to do that we changed the design of the experiment. We moved from a simple randomized trial to a balanced crossover design. This has the advantages of reducing the influence of confounding covariates (e.g., participant's skills), and requiring less participants with respect to non-crossover designs to achieve the same power. Conversely, crossover designs might suffer from the order in which the treatments are applied, or from carry-over and learning effects [43]. Using the experiment definition template in Wohlin et al. [45], this replication is defined as follows:
"Analyze the practice of TDD, for the purpose of evaluating its impact, compared to a test-last development approach, with respect to testing effort, external software quality and developers' productivity, from the point of view of the researcher, in the context of an academic course about software development practices."

4.2 Interaction between researchers

The interaction between the baseline experimenters (UOULU: authors DF, BT, and MO) and the replication ones (UNIBAS: authors GS and SR) followed the guidelines proposed in [24]. The interaction was started by UOULU. An initial meeting between UNIBAS and UOULU happened face-to-face, the baseline experiment (e.g., its motivation, design choice, and results) and the possibilities for a replication were discussed. Subsequently, a series of meetings (i.e., adaptation meetings [24]) was carried out via telco to discuss important points such as: where to sample the participants from, the participants' type, the time frame available for the training and the experiment, the schedule of the replication, and the physical settings in which the replication would take place. The design (see Section 4.7) proposed for this replication seemed to fit the settings. The finals decision regarding the replication design was made at this stage. After the meetings it was clear that the context would be close to the UOULU one, with two main differences. The UNIBAS settings allowed *a)* all the participants to work individually (i.e., without pair-programming) due to the larger availability of computer rooms; *b)* because of the new design, an additional task should be used. The expected impact of the changes (see Section 4.3) were also discussed at this stage. The practicalities—e.g., the training material, the specific experimental tasks, questionnaires for the participants—were defined via email discussions. During the execution of the experiment no major inquiries or clarifications were necessary. Finally, after the the analysis of the data was made available (see Section 4.8), there was a final combination meeting [24] via telco. Rather than focus strictly on the results, this meeting reviewed the changes on replication condition and mapped them onto the results.

4.3 Changes to original settings

With respect to UOULU, there were changes in the experimental protocol [19].
Experimental Design: the overall design was kept as one factor, two treatment but a second measurement—corresponding to the second task—was added, which resulted in a balanced crossover design. A balanced crossover design is a type of repeated measure design—i.e., the measures are taken several times for the same participant—in which a participant is randomly assigned to a sequence of treatments rather than a single one. That is, a participant will be assigned either to the sequence TDD+TLD, or to TLD+TDD. The expected

Table 2: UNIBAS hypotheses and outcomes

Name	H_0	H_1
R1T	$TEST(TDD) = TEST(TLD)$	$TEST(TDD) \neq TEST(TLD)$
R1Q	$QLTY(TDD) = QLTY(TLD)$	$QLTY(TDD) \neq QLTY(TLD)$
R1P	$PROD(TDD) = PROD(TLD)$	$PROD(TDD) \neq PROD(TLD)$

impact of this change is to improve the reliability of the results, i.e., a crossover design removes the within-participant difference since each participant serves as its own control. In particular, our concern regarded the participants' programming skill-set as a possible confounding covariate [17, 31]. A second change in the experimental design was done. In particular, UOULU had to include an additional nuisance factor, pair-programming, due to its settings. The effects of the nuisance factor were managed by an ad-hoc analysis. The settings of UNIBAS removed the need to control for such factor entirely. The expected impact of this change is that we can exclude the possibility that the results were due to the interaction with the nuisance factor.

Experimental Objects: following from the change in the experimental design, a change in the experimental objects' population was needed. We added a second object, MarsRover Api (MRA)—an experimental task, assessed by the authors as complex as the original one. This was a necessary change to implement the crossover design. In fact, using the same task for the second measurement would pose a significant threat to the internal validity of the experiment due to learning effects. In other words, there was the risk that the participants would perform better not because of the treatment but simply because they learned the task's solution during the first measurement. Given the changes described in this section, we consider this to be a close replication [38].

4.4 Research Questions and Hypothesis

As described in section 4.3, the research questions tackled in this replication are the same as the UOULU experiment. Nevertheless, their direction was changed accordingly. In particular, no expectation about the direction of the results is present. The research questions are formulated as follows:

RQ1: Is there a difference between the number of tests written by TDD developers and TLD developers?

RQ2: Is there a difference between the external code quality of TDD developers and TLD developers?

RQ3: Is there a difference between the productivity of TDD developers and TLD developers?

Table 2 reports the research hypothesis investigated in this replication, where $TEST$ is number of tests, $QLTY$ is external quality, and $PROD$ is developers' productivity.

4.5 Sample and Participants

The participants in UNIBAS were sampled by convenience among the students enrolled in the Informative System (IS) course of the Master program in Computer Science at the University of Basilicata. Students participated in the experiment on a voluntary basis. Although they were informed about their participation to the experiment, our research questions were not disclosed to them. We also informed participants that data were treated anonymously, and disclosed in aggregated form. Out of the 23 participants enrolled in the IS course, 21 decided to take part in the experiment. The course had elements regarding software testing, software development process, software maintenance, agile development techniques with a focus on TDD, regression testing, and refactoring. The participants had sufficient level of technical maturity and knowledge of software design, development, and refactoring. Their prior knowledge could be considered homogeneous. The participants had passed all the exams related to the following courses: Procedural Programming, Object-Oriented Programming I and II, Software Engineering I, Web Technologies, and Databases. In these courses, participants gained significant experience with C/C++, C#, and Java. The results of a pre-questionnaire administered before the experiment to the participants, showed that they had an average experience of 4.5 years in object oriented programming in academia (standard deviation 1.1 years). One participant declared ten years of professional experience in software development; three declared four, three and two years experience as professional developers; two participants declared one year of professional experience, while the remaining had none. The participants' average experience with unit testing was a little over one and a half years. The replication was conducted as an optional exercise of IS. Before the replication, the IS course was accompanied by four lab sessions of three hours each. During these lab sessions, students improved their knowledge on how to use the Eclipse IDE to develop unit tests with Java and JUnit, the refactoring tools included in the IDE, and TDD development. The participants were required to use TDD to solve several code katas of increasing difficulty throughout these lab sessions. We also asked students to work on home assignments to further practice the contents presented in the lab sessions. The topics, focusing on TDD, and material used during the training (i.e., slides, practice tasks, pre-questionnaire) was the same used for the UOULU experiment and translated into the Italian language.[6]

4.6 Objects

Two experimental tasks, one for each measurement, were used in this replication due to the changes in the experimental design. The first task was MarsRover API (MRA). It required to develop the API which allows to move a vehicle on a two-dimensional grid. It was algorithmic in nature, and several edge cases needed to be handled. The requirements were described as 11 user stories, which were later used for the acceptance test suite necessary to calculate the metrics. The participants were given a template project of the task to get started. It included the class containing the signature (8 LOC) and a stub of a JUnit test class (9 LOC). The second object was BSK, also used for UOULU.

4.7 Design, Context and Execution

The design is shown in Table 3. The experiment took place in two different days: April 15th (i.e., Day 1) and April 22nd 2015 (i.e., Day 2). We randomly assigned the participants to a sequence of treatments. Sequence A implemented MRA during Day 1 using TDD, and BSK during Day 2 using TLD. On the other hand, Sequence B implemented MRA during Day 1 with TLD, and BSK during Day 2 using TDD. In this design the treatments are switched within the sequence so that all the participants are given all the treatment yielding a balanced design. This not only removes within participants variance, but also controls for learning effects, as the participants do not apply the the treatments to the same task. The control group is formed by the participants applying TLD (Sequence A, Task BSK and

[6]DF is the author of the material and native Italian speaker.

Table 4: UOULU and UNIBAS settings

Characteristic	Baseline	UNIBAS
Participant type	33 graduate, 25 undergraduate	21 graduate students
Participant Unit	11 pairs, 36 individuals	21 individuals
Environment	Software quality and testing course	Informative system
Participant Stratification	None	None
Programming Environment	Java, Eclipse, JUnit	Java, Eclipse, JUnit
Experiment Task	Bowling Scorekeeper	Mars Rover API and Bowling Scorekeeper
Task Type	Fine grained, incremental difficulty	Fine grained, incremental difficulty
Time to Complete the Task	Single lab session (3 hours)	Two lab sessions (3 hours each)
Experiment Design	Parallel one factor, two treatments	Balanced crossover design
Experiment groups	TLD vs. TDD	TLD vs. TDD
Control / Treatment size	20 (4 pairs, 16 individuals) / 27 (7 pairs, 20 individuals)	21/21
Variables	$TEST$, $PROD$, $QLTY$, $SUBJ$	$TEST$, $PROD$, $QLTY$

Table 3: Replication design

		TASK	
		Day 1 (MRA)	Day 2 (BSK)
SEQUENCE	A	TDD	TLD
	B	TLD	TDD

Sequence B, Task MRA). The experimental group is formed by the participants applying TDD (Sequence A, Task MRA and Sequence B, Task BSK). Consistently with UOULU, *development approach* is the independent variable (IV). It is a nominal variable and assumes values: TLD and TDD. The instructions given to participants in both groups on how to apply the treatments during the experiment sessions were the same as UOULU. The dependent variables (DV): testing effort ($TEST$), external quality ($QLTY$), and productivity ($PROD$) are calculated as the UOULU experiment, using the same acceptance test suite for BSK. A new acceptance test suite was developed to measure $QLTY$, and $PROD$ for MRA. The context of the experimental sessions was the same as UOULU. Therefore, Java was the language used to solve the task, Eclipse was the IDE, three hours were allocated for the session. Just before the experiment, the experiment supervisors highlighted the study goal without providing details on its hypotheses. The experimental tasks were carried out under controlled conditions. GS and SR monitored the participants to prevent their communication with each other in the laboratory trials. The experiment supervisors were the same at each trial. We suggested the participants to implement the stories in the order with which they appear in the given documentation. We allowed all the participants to use Internet to accomplish the tasks because actual developers usually exploit this medium as support for their daily work activities. We clearly forbid the participants to use of the Internet to communicate with each another. Table 4 compared the settings of both experiments.

4.8 Blind analysis

In order to reduce possible researcher bias [26], we used a two-step protocol for blind analysis: *1)* blind data extraction (UPM); *2)* blind statistical analysis (BRUNEL). In 1), the source code produced during the experiment was given to a third party (i.e., author FU) to extract the metrics for the constructs of interest using the formulas reported in Section 3.4. FU used the same artifacts—i.e., metrics extraction form, and acceptance test suite—utilized for the data extraction in UOULU, as well as the acceptance test suite for the new task. He was aware of neither the experimental motivations, hypotheses, nor design. When extracting the data, one artifact was discarded because it did not contained the source code necessary to executed the acceptance test suite. The result of the first step was the dataset, comprising 20 observations, later used to carry out the statistical analysis in step 2). The dataset was then blinded by removing the labels from its column, so that, for the analyzer, the measurements do not refer to any particular construct. The blind analyzers (i.e., authors BS and MS) received the blinded dataset with a description of the variables (e.g., continuous, ordinal), their ranges, and the description of the experimental design in which the names of the constructs, and tasks were replaced with anonymous labels. The result of the second step was a document, containing the statistical analysis, upon which the results are based. Before making any inference on the results, the original labels were re-applied to them. In Figure 1, we summarize the interactions between the sites participating in this replication.

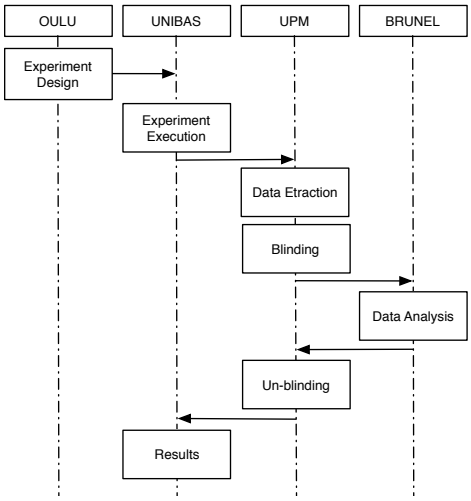


Figure 1: Sites involved in the replication

4.9 Results

In this section we present a summary of the statistical analysis process that led to the results.

4.9.1 Descriptive statistics

Table 5 presents the summary statistics for the dataset. We have 40 observations (i.e., two observations for each of the 20 valid participants) for the three DVs, $TESTS$,

$QLTY$, and $PROD$. It seems that the participants were able to correctly complete a bit more than one-third of the tasks ($\overline{PROD} = 39.81$); however the quality of the portions of the tasks implemented was acceptable ($\overline{QLTY} = 66.70$). It appears that the participants, on average, wrote less than two unit tests per each user story[7].

Table 5: Summary statistics for the UNIBAS dataset (n=40)

DV	Mean	St. Dev.	Min	Median	Max
TESTS	10	6.58	0	10	26
QLTY	66.70	30.77	0.00	74.35	100
PROD	39.81	33.05	0.00	35.79	100

4.9.2 Comparison of treatments

Figure 2 shows the distributional characteristics of the two treatments for each of the three DVs. The three research hypotheses $R1T$, $R1Q$, and $R1P$ were tested with $\alpha = .05$ as in UOULU using a non-directional one-sample sign test:

H_0: $M(DV_{(TDD)} - DV_{(TLD)}) = 0$
H_1: $M(DV_{(TDD)} - DV_{(TLD)}) \neq 0$

where M is the median, and $DV \in \{TESTS, PROD, QLTY\}$. The test results, reported in Table 6, show that none of the null hypotheses could be rejected. Moreover, the effect size [12]—reported as Cliff's δ—is negligible for all the variables. We conclude that no difference between TDD and TLD could be observed in terms of testing effort, external code quality, and developers' productivity. These results are in line with those of UOULU.

Table 6: Results of One Sample Sign test, n = 20, no ties were present

Hypotheses	p-value	Cliff's δ (CI)	Decision
R1T	1.00	0.07 (-0.41, 0.29)	Failed to reject H_0
R1Q	0.82	-0.07 (-0.29, 0.41)	Failed to reject H_0
R1P	0.82	0.14 (-0.22, 0.47)	Failed to reject H_0

4.9.3 Ordering Effects

The analysis included a second level of investigation initiated by BRUNEL, in which the sequence of treatments was conjectured to be a moderator variable. Hence, the question of whether the DV median's difference significantly varies between the two sequences A and B was investigated. The Kruskal-Wallis (K-W) test [43], i.e., a non-parametric tests for assessing whether two or more samples originate from the same distribution, was used. The hypotheses are:

H_0: $M(DV_{(TDD)} - DV_{(TLD)})_A = M(DV_{(TDD)} - DV_{(TLD)})_B$
H_1: $M(DV_{(TDD)} - DV_{(TLD)})_A \neq M(DV_{(TDD)} - DV_{(TLD)})_B$

Table 7 shows the hypotheses testing results. There is evidence of a difference between the sequences in all three DVs. We attribute this result to a *carryover effect*, or a *bias in the task*. The next section analyzes possible carryover effects.

4.9.4 Carryover Effects

After acknowledging the results of the blind analysis reported in Section 4.9.1-4.9.3, we suspected that a carryover might exist. A carryover effect takes place when the effect of the first treatment is still present when measuring the second. In particular, we have reasons to believe that there could be a carryover in the sequence B. In fact, applying TLD first might better prepare the participants for the application of TDD. For example, by using TLD as first treatment, the participants can focus on tackling small part of the task without the burden of applying the counterintuitive test-first approach at the same time. Conversely, applying TDD as first treatment, might overload the participants. Although the participants should be familiar with both techniques, we proceed with a carryover analysis to remove any doubts. When assessing the carryover effect, we followed the method proposed in [44]—i.e., a carryover is present when there is a statistically significant difference between the sums of the DVs at different experimental conditions with respect to the sequence. In order to assess such differences, we used the same statistical test used during the blind analysis:

H_0: $M(DV_{(TDD)} + DV_{(TLD)})_A = M(DV_{(TDD)} + DV_{(TLD)})_B$
H_1: $M(DV_{(TDD)} + DV_{(TLD)})_A \neq M(DV_{(TDD)} + DV_{(TLD)})_B$

where $DV \in \{TESTS, PROD, QLTY\}$.
The null-hypotheses were failed to reject (see Table 8); therefore we do not have strong evidence that our data exhibits a carryover effect. However, at this stage we cannot be sure whether the ordering effect showed is due to the task—i.e., the particular sequence of tasks is biased towards (or against) one sequence of treatments.

Table 7: K-W tests for ordering effects (n=20, df=1)

DV	K-W χ^2	p-value	Decision
TESTS	7.43	$.64 \times 10^{-2}$	Reject H_0
QLTY	8.69	$.32 \times 10^{-2}$	Reject H_0
PROD	9.14	$.24 \times 10^{-2}$	Reject H_0

Table 8: K-W tests for carryover effect (n=20, df=1)

DV	K-W χ^2	p-value	Decision
TESTS	1.22	.27	Failed to reject H_0
QLTY	.09	.76	Failed to reject H_0
PROD	.02	.88	Failed to reject H_0

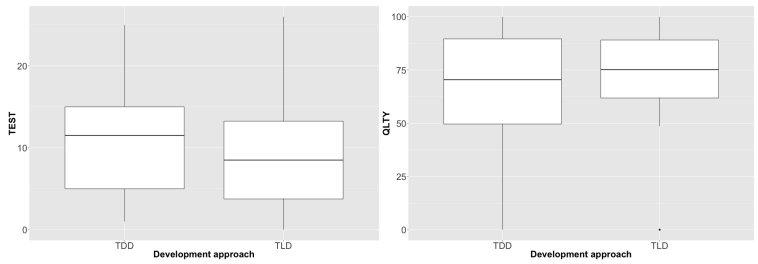
5. THREATS TO VALIDITY

We present and discuss threats that could affect our study. Despite our efforts to mitigate as many threats as possible, some of them are unavoidable. We discuss the threats to validity using the guidelines by Wohlin *et al.* [45]. Considering that this study focuses on testing a theory, rather than assess its generalizability, the threats to validity are prioritized in decreasing order of importance [45].

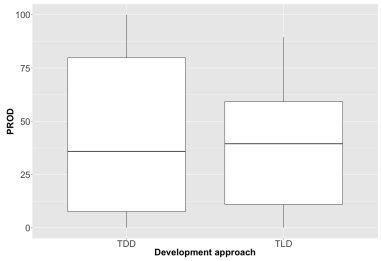
5.1 Internal Validity

Selection-maturation interaction: the adopted experimental design might suffer from the presence of carryover. We discussed it in Section 4.9.4. *Diffusion or imitation of treatments*: this threat concerns information exchanged among the participants within each trial. Experiment supervisors monitored the participants to prevent their communication to each another. As an additional measure to prevent diffusion of material, we asked participants to return back material at the end of each task. *Selection*: the effect of letting

[7]Note that number of total user stories is different between the two tasks (MRA=11, BSK=13).



(a) Difference in TEST (b) Difference in QLTY



(c) Difference in PROD

Figure 2: Boxplots comparing the DVs for the treatments in UNIBAS

volunteers take part in the experiment may influence the results, since they are generally more motivated [45].

5.2 External Validity

Interaction of selection and treatment: the use of students as participants may affect the generalizability of the results with respect to practitioners [8, 20]. However, the tasks to be performed did not require a high level of industrial experience, so we believe that the use of students as participants could be considered appropriate, as suggested in [5]. *Interaction of setting and treatment*: In our study, the size and the complexity of the selected application might affect external validity. The use of Eclipse might have threaten the result validity; it could be possible that a few participants in the experiment were more familiar with the tool than others.

5.3 Construct Validity

Interaction of different treatments: To mitigate this kind of threat, we adopted the design shown in Table 3. *Mono-method bias*: we acknowledge that this replication suffers from such bias since the DVs were measured with a single metric. However, we designed this replication to be as close as possible to the baseline. *Evaluation apprehension*: we mitigated this threat because the participants were not evaluated on the results they achieved during the experiment. The participants were also unaware of the objectives and the experimental hypotheses. *Experimenters' expectancies*: experimenters can bias results both consciously and unintentionally based upon their expectations about the outcome of the experiment. We mitigated this kind of threat by adopting a blind analysis procedure. *Reliability of measures*: we used measures that limit the influence of the measurer. The majority of the measures were automatized, and can be repeated with the same outcome. When human judgement was necessary, we adopted a blind approach.

5.4 Conclusion Validity

Random heterogeneity of participants: our sample includes participants with similar background—i.e., students in the same university who completed similar courses, and had similar experience. *Fishing and the error rate*: fishing has been mitigated because we applied a blind data analysis procedure. As for error rate, we chose an adequate significance level while testing null hypotheses. *Violated assumptions of statistical tests*: to perform certain tests some assumptions have to be verified (e.g., normality of data). Violating the assumptions may lead to wrong conclusions. We used robust, non-parametric statistics to test our hypotheses.

6. DISCUSSION AND COMPARISON

In this section we discuss and compare the results of the UOULU experiment with the UNIBAS replication. We discuss the current state of knowledge about the effects of TDD according to the changes that were made in the replication. Regarding the three research questions driving this replication ($RT1$, $RQ1$, and $RP1$), the answer is that, given the data collected during the experiment, there is no evidence supporting the claim that TDD differs from a TLD approach in terms of testing effort, external code quality, and developers' productivity. This replication yielded only consistent results [6], in similar settings, but a different design, and with different experimenters. Thus, the baseline experiment results are less likely to be a false negative, and less likely to be due to a bias of the baseline experimenters. The modifications to the experiment have not changed the results; therefore increasing the conclusion validity of the baseline experiment. We discuss the following changes.

Crossover design: In the baseline experiment, the possible impact of the difference between the participants in terms of skills was considered a threat [29]. The replication design decreases the possibility that the baseline experiment results were due to an unobserved, unbalanced skills distribution.

Pair-programming: In the baseline experiment, pair programming was used together with the treatments, and its effects assessed; the operationalization of the constructs was

improved in this replication by removing pair-programming. We are more confident that the baseline results were not due to the interaction with the other practice.

Task: The crossover design required a new task. We added a task of the same difficulty and with the same structure of the baseline experiment task. Although we believed that such change would not directly impact the baseline experiment results, there is evidence that it might interact with the sequence in which the treatments were applied.

Given the limitations presented in Section 5, it appears that TDD does not improve, nor deteriorate the participants' performance with respect to an iterative development technique in which unit tests are written after production code. The same conclusion has been reached in the relevant literature [32,36,41]. We conjecture that the inconclusive findings can be due to two factors:

Control treatment: the baseline experiment and its replication compared TDD to a really similar approach, labelled as TLD. Under this circumstances, we might be focusing on the incorrect part of development process (i.e., whether write tests first or not), and disregard the part of the process in which the a substantial effect might lie (i.e., the *iterativeness* of the process). Accordingly, the tasks used for both experiments were designed to fit the iterative nature of both treatments—i.e., isolate the process itself from the cognitive effort required to break down a complex problem into subproblems. Pančur and Ciglarič [33] made a similar claim reporting the inconclusive results of a similar experiment.

Tasks sequence: although the baseline experiment used a single task, this replication included a second task deemed of similar complexity. Since we showed a difference between the sequences, but not a carryover effect, there is the possibility that the (fixed) sequence of the experimental tasks interacts with one (or both) treatment sequence—for example, the MRA task inherently suits TLD, while BSK tends to be TDD's sweet-spot, or vice versa. We are worried about how experimental tasks should be chosen (or designed), not only to be comparable, but also to limit their interaction with the treatments, for crossover design studies [42].

7. CONCLUSION AND FUTURE WORK

In this paper we reported a replication of an experiment [13] in which TDD was compared to a test-last approach. The replication involved four institutions in four countries. In particular, data extraction and analysis were conducted by researchers unaware of the replication goals and hypotheses. Despite adopting such countermeasures, aimed at reducing researchers bias, we confirmed the baseline results: TDD does not affect testing effort, software external quality, and developers' productivity. The changes introduced in the replication did not affect the results—including the new design, crossover, for which no significant carryover effect was found. However, we found that there is a difference between the sequences in which the treatments were applied. We made sure to select similar experimental objects; nonetheless, we suspect that they interact with the treatments. We recommend future studies to survey the tasks used in experiments evaluating TDD, and assess them with respect to the treatments. Finally, in order to lower the cost of replication studies and reduce researchers' bias, we encourage other research groups to adopt the same multi-site approach described in this paper.

Acknowledgments

This research is supported in part by the Academy of Finland Project 278354.

8. REFERENCES

[1] M. Baker. Over half of psychology studies fail reproducibility test. *Nature News*, 27, 2015.

[2] M. T. Baldassarre, J. Carver, O. Dieste, and N. Juristo. Replication types: Towards a shared taxonomy. In *Proc. of International Conference on Evaluation and Assessment in Software Engineering*, pages 18:1–18:4. ACM, 2014.

[3] V. Bebarta, D. Luyten, and K. Heard. Emergency medicine animal research: does use of randomization and blinding affect the results? *Academic Emergency Medicine*, 10(6):684–687, 2003.

[4] K. Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[5] J. Carver, L. Jaccheri, S. Morasca, and F. Shull. Issues in using students in empirical studies in software engineering education. In *Proceedings of International Symposium on Software Metrics*, pages 239–251, 2003.

[6] J. C. Carver. Towards reporting guidelines for experimental replications: A proposal. *Proceedings of the 1st international workshop on replication in software engineering (RESER)*, 2010.

[7] M. Ciolkowski. What do we know about perspective-based reading? an approach for quantitative aggregation in software engineering. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 133–144. IEEE Computer Society, 2009.

[8] M. Ciolkowski, D. Muthig, and J. Rech. Using academic courses for empirical validation of software development processes. *EUROMICRO Conference*, pages 354–361, 2004.

[9] F. Q. B. da Silva, M. Suassuna, A. C. C. França, A. M. Grubb, T. B. Gouveia, C. V. F. Monteiro, and I. E. dos Santos. Replication of empirical studies in software engineering research: a systematic mapping study. *Empirical Software Engineering*, 19(3), 2014.

[10] H. Erdogmus, M. Morisio, and M. Torchiano. On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(3):226–237, Mar. 2005.

[11] M. Fowler. *Refactoring: improving the design of existing code*. Pearson Education India, 2009.

[12] C. O. Fritz, P. E. Morris, and J. J. Richler. Effect size estimates: current use, calculations, and interpretation. *Journal of Experimental Psychology: General*, 141(1):2, 2012.

[13] D. Fucci and B. Turhan. A Replicated Experiment on the Effectiveness of Test-First Development. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 103–112. IEEE, Oct. 2013.

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

[15] D. Fucci, B. Turhan, and M. Oivo. Conformance

factor in test-driven development: initial results from an enhanced replication. In *Proc. of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014.

[16] D. Fucci, B. Turhan, and M. Oivo. Impact of process conformance on the effects of test-driven development. In *the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurements*, pages 1–10. ACM Press, 2014.

[17] D. Fucci, B. Turhan, and M. Oivo. On the effects of programming and testing skills on external quality and productivity in a test-driven development context. In *Proc. of the 19th International Conference on Evaluation and Assessment in Software Engineering*, EASE '15. ACM, 2015.

[18] B. George and L. Williams. A structured experiment of test-driven development. *Information and Software Technology*, 46(5):337–342, 2004.

[19] O. S. Gómez, N. Juristo, and S. Vegas. Understanding replication of experiments in software engineering: A classification. *Information and Software Technology*, 56(8):1033–1048, Aug. 2014.

[20] J. Hannay and M. Jørgensen. The role of deliberate artificial design elements in software engineering experiments. *IEEE Trans. on Soft. Eng.*, 34:242–259, March 2008.

[21] T. P. Hettmansperger and J. W. McKean. *Robust nonparametric statistical methods*. CRC Press, 2010.

[22] M. Ivarsson and T. Gorschek. A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16(3):365–395, Oct. 2010.

[23] N. Juristo and S. Vegas. Using differences among replications of software engineering experiments to gain knowledge. In *Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on*, pages 356–366. IEEE, 2009.

[24] N. Juristo, S. Vegas, M. N. Solari, and S. Abrahão. A process for managing interaction between experimenters to get useful similar replications. *Information and Software Systems Journal*, 2013.

[25] B. Kitchenham. The role of replications in empirical software engineering - a word of warning. *Empirical Software Engineering*, 13(2):219–221, 2008.

[26] R. MacCoun and S. Perlmutter. Blind analysis: Hide results to seek the truth. *Nature*, 526:187–189, 2015.

[27] L. Madeyski. The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment. *Information and Software Technology*, 52(2):169–184, 2010.

[28] E. M. Maximilien and L. Williams. Assessing test-driven development at ibm. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 564–569, Washington, DC, USA, 2003. IEEE Computer Society.

[29] S. E. Maxwell and H. D. Delaney. *Designing experiments and analyzing data: A model comparison perspective*, volume 1. Psychology Press, 2004.

[30] M. G. Mendonça, J. C. Maldonado, M. C. F. de Oliveira, J. Carver, S. C. P. F. Fabbri, F. Shull, G. H. Travassos, E. N. Höhn, and V. R. Basili. A framework for software engineering experimental replications. In *Proc. of International Conference on Engineering of Complex Computer Systems*, pages 203–212. IEEE Computer Society, 2008.

[31] M. M. Müller and A. Höfer. The effect of experience on the test-driven development process. *Empirical Software Engineering*, 12(6):593–615, 2007.

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

[33] M. Pančur and M. Ciglarič. Impact of test-driven development on productivity, code and tests: A controlled experiment. *Information and Software Technology*, 53(6):557–573, June 2011.

[34] K. Pearson. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242, 1895.

[35] S. L. Pfleeger and W. Menezes. Marketing technology to software practitioners. *IEEE Software*, 17(1):27–33.

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

[37] M. Shepperd, D. Bowes, and T. Hall. Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, 40(6):603–616, 2014.

[38] F. Shull, J. C. Carver, S. Vegas, and N. J. Juzgado. The role of replications in empirical software engineering. *Empirical Software Engineering*, 13(2):211–218, 2008.

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

[40] B. Sigweni and M. Shepperd. Using blind analysis for software engineering experiments. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, page 32, 2015.

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

[42] S. Vegas, C. Apa, and N. Juristo. Crossover designs in software engineering experiments: Benefits and perils. *IEEE Transactions on Software Engineering*, 42(2):120–135, Feb 2016.

[43] E. Vonesh and V. M. Chinchilli. *Linear and nonlinear models for the analysis of repeated measurements*. CRC press, 1996.

[44] S. Wellek and M. Blettner. On the proper use of the crossover design in clinical trials. *Deutsches Arzteblatt Intern*, 109:276–281, 2012.

[45] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.