*Title Page containing ALL Author Contact Info.

Noname manuscript No.
(will be inserted by the editor)

On The Role of Tests in Test-driven Development: A Differentiated Replication

Davide Fucci · Burak Turhan

the date of receipt and acceptance should be inserted later

Abstract Background: Test-Driven Development (TDD) is claimed to have positive effects on external code quality and programmer's productivity. The main driver for these possible improvements is the tests enforced by the test-first nature of TDD as previously investigated in a controlled experiment (i.e. the original study).
Aim: Our goal is to examine the nature of the relationship between tests and external code quality as well as programmers' productivity in order to verify/ refute the results of the original study.
Method: We conducted a differentiated and partial replication of the original setting and the related analyses, with a focus on the role of tests. Specifically, while the original study compared test-first vs. test-last, our replication employed the test-first treatment only. The replication involved 30 students, working in pairs or as individuals, in the context of a graduate course, and resulted in 16 software artifacts developed. We performed linear regression to test the original study's hypotheses, and analyses of covariance to test the additional hypotheses imposed by the changes in the replication settings.
Results: We found significant correlation (Spearman coefficient=0.66, with p-value=0.004) between the number of tests and productivity, and a positive regression coefficient (p-value=0.011). We found no significant correlation (Spearman coefficient=0.41 with p-value= 0.11) between the number of tests and external code quality (regression coefficient p-value=0.0513). For both cases we observed no statistically significant interaction caused by the subject units being individuals or pairs. Further, our results are consistent with the original study although there were changes in the timing constraints for fin-

Davide Fucci
Department of Information Processing Science, University of Oulu
E-mail: davide.fucci@oulu.fi

Burak Turhan
Department of Information Processing Science, University of Oulu
E-mail: burak.turhan@oulu.fi

ishing the task and the enforced development processes.
Conclusions: This replication study confirms the results of the original study concerning the relationship between the number of tests vs. external code quality and programmer productivity. Moreover, this replication allowed us to identify additional context variables, for which the original results still hold; namely the subject unit, timing constraint and isolation of test-first process. Based on our findings, we recommend practitioners to implement as many tests as possible in order to achieve higher baselines for quality and productivity.

Keywords Test-driven development, software quality, productivity, software testing, replication

Noname manuscript No.
(will be inserted by the editor)

On The Role of Tests in Test-driven Development: A Differentiated and Partial Replication

Davide Fucci · Burak Turhan

the date of receipt and acceptance should be inserted later

Abstract Background: Test-Driven Development (TDD) is claimed to have positive effects on external code quality and programmers' productivity. The main driver for these possible improvements is the tests enforced by the test-first nature of TDD as previously investigated in a controlled experiment (i.e. the original study).
Aim: Our goal is to examine the nature of the relationship between tests and external code quality as well as programmers' productivity in order to verify/ refute the results of the original study.
Method: We conducted a differentiated and partial replication of the original setting and the related analyses, with a focus on the role of tests. Specifically, while the original study compared test-first vs. test-last, our replication employed the test-first treatment only. The replication involved 30 students, working in pairs or as individuals, in the context of a graduate course, and resulted in 16 software artifacts developed. We performed linear regression to test the original study's hypotheses, and analyses of covariance to test the additional hypotheses imposed by the changes in the replication settings.
Results: We found significant correlation (Spearman coefficient=0.66, with p-value=0.004) between the number of tests and productivity, and a positive regression coefficient (p-value=0.011). We found no significant correlation (Spearman coefficient=0.41 with p-value= 0.11) between the number of tests and external code quality (regression coefficient p-value=0.0513). For both cases we observed no statistically significant interaction caused by the subject units being individuals or pairs. Further, our results are consistent with the original study although there were changes in the timing constraints for fin-

Davide Fucci
Department of Information Processing Science, University of Oulu
E-mail: davide.fucci@oulu.fi

Burak Turhan
Department of Information Processing Science, University of Oulu
E-mail: burak.turhan@oulu.fi

ishing the task and the enforced development processes.
Conclusions: This replication study confirms the results of the original study concerning the relationship between the number of tests vs. external code quality and programmer productivity. Moreover, this replication allows us to identify additional context variables, for which the original results still hold; namely the subject unit, timing constraint and isolation of test-first process. Based on our findings, we recommend practitioners to implement as many tests as possible in order to achieve higher baselines for quality and productivity.

Keywords Test-driven development, software quality, productivity, software testing, replication

1 Introduction

Test-Driven Development (TDD) is a programming technique that encourages code development by repeating short cycles consisting of [Beck, 2003]:

1. writing a test for an unimplemented functionality or behavior;
2. supplying the minimal amount of production code to make the test pass;
3. refactoring the production code;
4. checking that all tests are still passing after the changes.

These result in respectively: a shift from test-last to test-first approach; developing only enough code to pass the tests; focusing on design quality through refactoring; and a dynamic library of regression tests. Hence, it is claimed that TDD leads to better code quality and improves the developer confidence in the code enhancing her/his productivity [Astels, 2003].

A series of independent experiments contrasting TDD with test-last development has been carried out to assess the validity of such claims (see Section 2). The experiments compared the external code quality and/or programmer productivity of test-first and test-last approaches, but the overall view has been inconclusive [Turhan et al., 2010, Shull et al., 2010]. Pedroso et al. argue that trying to gauge the association between the development approach (TDD vs. non-TDD) and the code quality and/or productivity in a direct manner might be inadequate and ineffective to evaluate the effectiveness of TDD, or it could be at least hindered by hidden variables [Pedroso et al., 2010].

Our study focuses on the analysis of the role of tests as the main reason for the claimed effects, as opposed to studies comparing the external code quality and/or programmer productivity of test-first and test-last approaches. Indeed, previous research often tended to neglect the role of tests except Erdogmus et al., whose study is unique in this sense [Erdogmus et al., 2005]. As a matter of fact, tests written in a TDD fashion differ from the ones written in a typical test-last context, the main difference being the quantity of tests and the test-to-production code ratio [Beck, 2003, George and Williams, 2003]. Erdogmus et al. used a simple measurable characteristic of tests, i.e. the number of unit-tests, to predict for external code quality and productivity. A higher number of tests is foreseen as an endogenous symptom of code quality and critical

for quality assurance. From a productivity perspective, a higher number of tests is supposed to pay back in the long-term by enhancing the focus of the programmers through tests, though this seems counter-intuitive in the short-term.

This paper presents a partial, differentiated replication of Erdogmus et al.'s study in an academic setting with a focus on the investigation of the effects of tests on external code quality and programmer productivity. We should note that a comparison of test-first vs. test-last is also a part of the original study that we left out of the scope of this study. We only investigate the relationship between the number of tests, as a central component of the TDD approach, vs. external code quality and programmer productivity. Therefore, we consider this replication as being partial, and prefer to use the term "replicated study" instead of "replicated experiment". There were some changes in the study context and measurements, hence our study is a differentiated replication.

Differentiated replications are important for the empirical software engineering discipline, and they can help researchers improve the confidence in the results as well as identify which variables could affect the outcome [Juristo and Vegas, 2009]. As stated by Juristo & Vegas, researchers should "not be obsessed by identical replication" but strive to carry-out replications, even though non-exact, that will bring new pieces of knowledge to the field. It is important to understand which kind of replication the study is aiming at, and check that it remains consistent till the end. Furthermore, study results with low significance or small sample size are valuable since they can be combined with other studies using meta-analysis techniques [Dieste et al., 2010].

The remaining of the paper is organized as follows: Section 2 presents studies in academic contexts regarding TDD's effectiveness, and the framework for reporting the study in this paper. Section 3 presents context, goals, hypotheses and results of the original study. Section 4 provides the description of this replication, in terms of goals, context, data, hypothesis and methods and changes to the original study, while in Section 5 the threats to validity are discussed. Section 6 compares the original and replicated study results. Finally, Section 7 presents the conclusions and addresses future work.

2 Related Work and Background

To provide a broad view on the topic, in this section we first present a brief overview of empirical studies in academia, in which the context is similar to our study, concerning the effectiveness of TDD over test-last development. Table 1 presents a summary of the results of those studies. Then, we report the results of empirical studies aimed at understanding the effects of TDD on code quality and programmer productivity without a comparison to test-last development. Finally, we describe the replication framework used to report this study.

2.1 Studies comparing test-first vs. test-last development

Table 1 Summary of previous studies assessing TDD's effectiveness over test-last in academic settings

ID	Authors	Study Type	# Subjects	Subjects unit	External Quality	Productiv
1	Pančur [Pančur and Ciglarič, 2011]	Cont. Exp.	112	Pairs and Individuals	No diff.	No diff
2	Xu [Xu S, 2009]	Case study	8	Individuals	Better	Better
3	Bhadauria [Bhadauria, 2009]	Cont. Exp.	130	Pairs	—	Better
4	Huang [Huang, 2007]	Cont. Exp.	55	Teams	—	No diff
5	Yenduri & Perkins [Yenduri and Perkins, 2006]	Case study	18	Individuals	Better	Better
6	George [George, 2002]	Cont. Exp.	138	Pairs	Better	Worse
7a	Madeyski [Madeyski, 2010] (1)	Quasi Exp.	188	Pairs and Individuals	No diff	No diff
7b	Madeyski [Madeyski, 2010] (2)	Cont. Exp.	24	Individuals	No diff	No diff
7c	Madeyski [Madeyski, 2010] (3)	Cont. Exp	27	individuals	No diff	No diff

In the study by Pančur and Ciglarič, based on four controlled experiments in academic settings, the authors report that TDD does not improve nor worsen external quality measured by the percentage of acceptance tests passed (PAPT), neither it seems to influence programmers productivity measured as the number of implemented user stories per hour (NIUSPH) [Pančur and Ciglarič, 2011]. On the other hand, encouraging results arise from other studies where external quality, measured as the number of acceptance tests passed (NAPT), is improved by TDD [Xu S, 2009, Yenduri and Perkins, 2006, George, 2002].

Productivity seems to be improved by TDD when accompanied by pair programming, where productivity is measured through verbatim recall metric defined as "the ability of programmers to recall key words or key concepts that they were expected to learn while working on the programming task" [Bhadauria, 2009, Yenduri and Perkins, 2006]. An opposite result is reported by George, but in this case productivity is measured simply as the time to complete the task, therefore such result might be due to the increased initial development time typical of TDD [George, 2002]. Another study, by Huang, shows that the effect of test-first approach on productivity, calculated as LOC/person*hour, is not statistically significant, even though TDD programmers indicated that their productivity was improved [Huang, 2007].

Madeyski reports three experiments in academic settings with M.Sc. students in computer science and engineering [Madeyski, 2010]. The meta-analysis shows that external quality, measured through PAPT, does not improve or worsen when using TDD. The same is true for programmers productivity, gauged as the number of acceptance tests passed per hour (NATPPH). TDD methodology seems to have positive impact on internal code quality, in terms of mean coupling between object and weighted methods per class metrics. Madeyski also analyzes the effect of TDD in terms of test quality using two different metrics, mutation score indicator and branch coverage, and reports that TDD does not improve or worsen either of them.

The empirical evidence gathered from both industry and academia through a systematic review of studies published between 2000 and 2008 shows that the general picture about the effects of TDD is blurred [Turhan et al., 2010, Shull et al., 2010].[1] The 22 reviewed publications, containing 32 controlled experiments comparing test-first to test-last approaches report moderate evidence for improvement in external quality in favor of TDD, and inconclusive results for productivity. The authors' suggestion for practitioners is to not use TDD as a *panacea* but to reflect also on other factors, such as developers' experience and organizational culture, when adopting TDD.

Recently, Rafique and Misic performed a meta-analysis covering 37 controlled experiments in 25 published papers until February 2011, in which test-first is compared with test-last approaches in both industrial and academic contexts [Rafique and Misic, 2012]. [2] They performed subgroup analysis dividing the experiments' contexts according to the development process followed by the control groups (Waterfall vs. Iterative Test Last) and the presence of others XP practices (Agile inclusive vs. Agile exclusive). The overall results show that TDD modestly increases quality, while it has no effects on productivity. However, the subgroup analysis allowed to identify the task size as a variable that might obfuscate TDD's effects on quality. Moreover, the use of other agile practices, e.g. pair programming, seems to improve productivity but this claim was not formally assessed. When analyzing the subgroups academic vs. industrial settings, a large reduction of productivity took place in the latter due to the more time spent by the more experienced developers in testing and refactoring activities. The authors also report the common issues identified by their literature review regarding experiment designs involving TDD. In particular:

- Subjects need better training, and training activities should be reported in the experiment.
- Experiments in industrial settings should focus on long-term effects of TDD.
- The experimental task should be reported objectively.
- Process conformance should be taken into account and reported.

Following such recommendations is likely to facilitate both replication and meta-analysis studies.

2.2 Studies on TDD's effects on quality and productivity

Melnik and Maurer surveyed 240 Canadian university students at different levels, from undergraduate to Ph.D. students, about their perception of extreme programming practices including TDD [Melnik and Maurer, 2005]. Around 70% of the respondents agreed that TDD improved software quality (external quality) and sped up testing process (productivity), while 60% agreed that it

[1] This systematic review includes studies 4, 5, 6 and 7x from Table 1.

[2] This meta-analysis includes studies 1, 2, 5, 6 and 7x from Table 1.

also improved software design (internal quality). Finally, 55% declared that they used or were willing to use TDD to develop at least one of their course assignments. In the same study a positive correlation between students' preference towards the use of TDD and their age was found this could be explained by the higher level of discipline required to properly apply TDD. On the other hand, less experienced students reported that they were confused and their productivity was hindered by the practice.

Janzen and Saiedian carried out six controlled experiments (five in academic and one in industrial settings) comparing TDD to test-last development [Janzen and Saiedian, 2007]. The results of the surveys carried among 160 subjects, 130 novice and 30 experienced programmers, showed that the latter have a better opinion of TDD since 70% of them believed that TDD led to a correct solution in less time, along with the 90% who thought that it also produced fewer defects and better tested code. On the other hand, only 30% of less experienced programmers had the opinion that TDD lead to correctness and 65% did not believe it produced code with less defects or that was better tested.[3]

Flohr and Schneider, during a controlled experiment with 18 graduate level students using pair programming and comparing TDD with test-last development, asked the 10 TDD group subjects (five pairs) to answer a questionnaire after each of the four iterations of the experimental task, to gauge their perceptions of TDD [Flohr and Schneider, 2006]. The authors reported that 85% thought that TDD neither improved or worsened quality in respect to other development techniques, 70% answered that TDD neither sped up or slowed down the development while 20% said that it slowed down the development. Interestingly 73% declared that TDD helped them "not to write more code than necessary" which can be related to an improvement in productivity; the 63% declared that TDD helped them to "write code with less flaws" which can be related to an improvement in quality.

Finally, an action research study by Keefe et al. followed 19 students over a period of eighteen weeks, during which the subjects were attending a course about several extreme programming practices including TDD [Keefe et al., 2006]. The analysis of the subjects' reflection showed that none of the students, even the ones who passed the course with excellent scores, expressed a preference for TDD. The main reason appeared to be that the subjects did not see the value of writing tests before the actual implementation, while the subjects who performed worst did not see the value of writing tests in general. The subjects felt that TDD was too complex due to the difficulty of selecting what to test and what data to use for their tests. The authors recommended to introduce general testing concepts and TDD afterwards, along with the appropriate tools (e.g. JUnit).

[3] In this study, all subjects were given instructions on both approaches.

Table 2 Original study hypotheses and outcomes (from [Erdogmus et al., 2005]). TL represents the test-last group while TF corresponds to the test-first group.

Stage	Name	H_0	H_1	Outcome
1	1T	$\#TESTS(TF) = \#TESTS(TL)$	$\#TESTS(TF) > \#TESTS(TL)$	Reject H_0
1	1Q	$QUALITY(TF) = QUALITY(TL)$	$QUALITY(TF) > QUALITY(TL)$	Accept H_0
1	1P	$PROD(TF) = PROD(TL)$	$PROD(TF) > PROD(TL)$	Accept H_0
2	2Q	$QUALITY = \beta_0 + \beta_1 \times \#TESTS, \beta_1 = 0$	$QUALITY = \beta_0 + \beta_1 \times \#TESTS, \beta_1 \neq 0$	Accept H_0
2	2P	$PROD = \beta_0 + \beta_1 \times \#TESTS, \beta_1 = 0$	$PROD = \beta_0 + \beta_1 \times \#TESTS, \beta_1 \neq 0$	Reject H_0

2.3 Replication Framework

To carry out the replication presented in this work we use the conceptual framework devised by Juristo and Vegas [Juristo and Vegas, 2009]. The authors express three fundamental concepts:

- **Classification of replication**: This concept aims at identifying which kind of differences appear between the original study and its replication. According to this classification, replications can be divided into close replication and differentiated replication. The former occurs when all or almost all the conditions of the original study remain the same or undergo only slight variations in the replication, the latter occurs when a major variation of the original study conditions are deliberately planned in the replication.
- **Replication process**: The replication process consists in identifying the differences between the original and the replicated study and in making new hypotheses about the impact of possible changes. Once the replication is executed and the data is analyzed, it is necessary to interpret the replication results and compare them with the original results.
- **Generating knowledge from replication**: Finally, the new knowledge brought by the replication arises both from the planned changes and from possibly unintended changes that may occur during the replication process.

Following this framework, we classify our study as a differentiated replication, and structure the paper following the guidelines proposed by Carver in order to report the replication process and the acquired knowledge [Carver, 2010].

3 Description of the Original Study

The original study isolated test-first approach in the TDD process with the goal of comparing it with the traditional test-last approach in order to evaluate their effects on external code quality and programmers productivity. On a second level of analysis, the authors investigated the relation between the number of tests, external quality and subjects' productivity. In this study we only replicated Stage 2 of the original study.

3.1 Research questions

The research questions from the original study [Erdogmus et al., 2005] are provided below. The research questions are also converted into experimental hypotheses, as shown in Table 2 along with their outcomes.

1. Do test-first programmers write more tests than test-last programmers ? (Hypothesis 1T in Table 2).
2. Do test-first programmers produce higher quality than test-last programmers? (Hypothesis 1Q in Table 2).
3. Are test-first programmers more productive than test-last programmers? (Hypothesis 1P in Table 2).
4. Does a higher number of tests imply higher quality? (Hypothesis 2Q in Table 2).
5. Does a higher number of tests imply higher productivity? (Hypothesis 2P in Table 2).

3.2 Subjects

The experiment involved 35 third-year Computer Science students, out of which 11 dropped out before completing the task. The subjects attended an intensive Java course before taking part in the experiment, during the course they learned object oriented programming in Java, software system design and unit testing. The course lasted eight weeks. [Erdogmus et al., 2005]

3.3 Artifacts

The subjects were required to implement a version of Robert Martin's Bowling Scorekeeper exercise adapted for the purpose of incremental development [George and Williams, 2003]. The original problem was divided into smaller, incremental stories of different difficulties. However, the final product had a simple design consisting of few classes and required only a command-line interface without GUI. The original study authors developed a black box acceptance test suite consisting of 150 test methods and a total of 350 JUnit asserts. The acceptance test suite was later run on each of the subjects' artifacts to assess their external quality level. [Erdogmus et al., 2005]

3.4 Design

The experiment took place in a computer laboratory equipped with 40 personal computers connected to the Internet and with the Eclipse IDE and CVS client installed. The subjects were randomly assigned to one of the two groups (test-first or test-last) and asked to implement the assignment accordingly. They had the opportunity to implement the stories constituting the assignment during

the lab sessions or at home. Before starting working on the assignment, in the lab or at home, the subjects needed to check out the project from the CVS repository using their personal credentials. Before stopping working on the assignment, in the lab or at home, they had to commit the project to the CVS repository. The subjects in both groups had to develop the experiment task incrementally, since the stories were built on the top of each other with incremental difficulty. The subjects and related artifacts considered for the analysis were only the ones who checked in the final version of the assignment into the repository. Along with the execution of the experiment, the authors gave the subjects two questionnaires, one to assess their skills and experience and another to check the effort they put in the assignments, their conformance to the process and opinion regarding the techniques. The subjects were informed about being part of an experiment before the study started, but the experiment hypothesis were not revealed until its completion. The participation in the experiment was voluntary, and data was collected only from subjects who signed an informed consent form. [Erdogmus et al., 2005]

3.4.1 Metrics

Erdogmus et al. used a defect-based metric to gauge external code quality (QLTY). The quality of each story delivered by the subjects (a story was considered delivered if it passed 50% of the acceptance tests) was calculated as the percentage of passing assertions in the acceptance test suite weighted by a difficulty factor associated with each test. The difficulty factor was based on the number of assert statements that constituted that particular acceptance test. The final score for a subject is given as the average of his stories' quality, therefore it ranges between 0.5 and 1. The metric used to measure productivity (PROD) was "output per unit effort", calculated as the number of delivered stories normalized by total programming effort. The total programming effort was estimated using both the log files from the CVS and the post-questionnaire data. The number of tests (TESTS) was normalized by total programming effort since the time spent by the subjects working on the assignment varied from few to 25 hours. Moreover, the authors filtered out useless tests such as empty or stub test methods and duplicated test methods. [Erdogmus et al., 2005]

3.5 Summary of results

The original study authors tested the hypotheses in Stage 1 using the Mann-Whitney U-test. Specifically, only in the case of 1T it was possible to reject the null hypothesis, while in the cases of 1Q and 1P the differences were not statistically significant. Therefore, the conclusion of the original study was that test-first programmers wrote more tests than test-last programmers, while nothing can be said about external quality nor about the level of productivity

Table 3 Stage 1 results of the original study (not included in this replication). The hypotheses in each row were tested comparing the values of metrics, i.e. TESTS, QLTY, PROD, described in Section 3.4.1.

Hypotheses	**Median**		**Mean**		**Std Dev**		***p*-value**
	Test-first	*Test-last*	*Test-first*	*Test-last*	*Test-first*	*Test-last*	
1T (TESTS)	0.90	0.45	1.31	0.86	1.07	1.00	0.09
1Q (QLTY)	0.84	0.92	0.83	0.85	0.11	0.14	0.25
1P (PROD)	0.67	0.55	1.09	0.89	1.05	0.71	0.48

among the two groups. Table 3 shows the descriptive statistics and p-values for Stage 1 of the original study.

To test Stage 2 hypotheses, the original authors used Spearman's correlation. In particular, hypothesis 2Q was not significant (p-value was not given) while hypothesis 2P was found to be statistically significant (p-value = 0.003 and Spearman coefficient = 0.587). Therefore, the original study found that "the number of tests can predict productivity independent from the group". The model was expressed by the equation *Productivity* = 0.255 + 0.659 * *Tests*, with an adjusted R^2 = 0.60 and a p-value, associated with the regression coefficient, of 0.001. The results remained statistically significant when the control group, composed of test-last programmers and the experiment group, composed of test-first programmers, were analyzed separately [Erdogmus et al., 2005]. Please refer to Table 2 for a summary of the original study results for both stages.

4 Replication

4.1 Replication motivation and goals

We aim at verifying/refuting the results of the original study and identifying the set of variables and conditions yielding those results. Following the original study, this replication leverages hypothesis 1T to check the validity of hypotheses 2Q and 2P (See Table 2).

The original study shows that a larger amount of tests are implemented when using the test-first approach to develop a fine-grained, incremental task, compared to the test-last approach. Moreover, Erdogmus et al. argued that the number of tests can be positively correlated with, and can be used to predict for external quality and productivity, yet found evidence only in favor of the latter regardless of the development method. Hence, we set our study in a test-first only setting in order to check the validity of this argument.

The goals of this replication are specified taking into account the hypothesis of Stage 2 of the original study, and are stated as follows:

> *Verifying/ Refuting* the effect of tests on external code quality and programmer productivity in an academic context, *through* a differentiated replication.

4.2 Interaction with the original investigators

The replication we carried out is considered to be external, since the original investigators were not involved directly. Nevertheless, the original investigators provided us a partial laboratory package [Brooks et al., 2008] consisting of the spreadsheets and graphs used to report their results, and the acceptance test suite used to verify the artifacts delivered by the subjects. The laboratory package was missing information about the rank of difficulty assigned to each user story implemented by the subjects as well as the original software artifacts delivered at the end of the study and the corresponding CVS log files. Moreover, we had a face-to-face discussion with the original investigators to ask feedback about our idea of partially replicating their study. The use we made of such material was to have a better understanding of the metrics used by the original investigators, and to check if it would be possible for us to adapt their acceptance tests suite to our study, but that was not the case (as described in Section 4.3.2). We are confident that the communication with the original investigators did not bias the execution of the replication.

4.3 Changes to the original study

Two types of changes were made compared to the original study. We report them below as: changes in the context and changes in the metrics.

4.3.1 Changes in the context

The following changes were made to the original study settings:

(a) Change in the settings, from TDD vs. Test-Last to TDD only.
(b) The time span required to implement the exercise was much shorter
(c) The Test-first aspect was not *formally* isolated
(d) Pair programming was encouraged

The change in point (a) was necessary since the scope of the course was focused on TDD. The impact of this change is that the replication of the original study had to be re-designed. Therefore, only a partial replication covering the second stage of Erdogmus et al. was possible.

The change in (b) was necessary and known before the study took place due to logistics reasons. The scope of the course was focused on TDD, and the lab rooms could only be booked for a short period of time. On the other hand, conducting the study in a single session in a computer laboratory allowed us tighter control on the environment compared to the original study, which permitted remote participation.

In the original experiment, the subjects tackled the task using a specific testing pattern they were taught by the experimenters, allowing to *"eliminate the role that low-level design plays in TDD, thereby isolating the Test-First aspect"* [Erdogmus et al., 2005]. This was not the case for our replication since

the objective of the course in which our study took place was to present TDD as a whole process, for instance including the role played by low-level design and refactoring, not only the Test-first approach *per se*, therefore the change in point (c) was necessary.

The change in point (d) was also necessary. The study was conducted in the scope of a course, and pair programming was one of the topics students were encouraged to practice. Nevertheless, this change enabled us to test for the effects of subgroups (i.e. pairs vs. individuals) on the overall results.

Table 4 provides a comparison of the context variables of the original study and this replication.

4.3.2 Changes in the metrics

In this replication the metrics used to gauge external code quality and productivity were changed. This was necessary due to the impossibility of getting or replicating the framework used for measuring the same variables in the original study. In other words, it was not possible to adapt the artifacts from this replication in order to comply with the acceptance test suite used in the original study. The original acceptance test suite included 105 black-box tests, while for this replication we were able to implement only a subset of it, containing 65 acceptance tests.

It must be noted that in the original study the number of user stories delivered by each subject was normalized by the total programming effort due to the variable time-frame to complete the task. In this replication, the amount of time was fixed, therefore that kind of normalization was not needed. Moreover, in the original study a story was considered delivered only *"if a story passed at least 50 percent of the assert statements from the associated acceptance test suite"* [Erdogmus et al., 2005]. In our study, the outcome of the acceptance test suite was not used to filter the stories implemented by the subjects. Thus, the level of productivity is expected to be relatively higher compared to the original study. Nevertheless, we were not able to obtain the raw, unnormalized data from the original investigators for a comparison. We do not expect this change to lead to any major impact.

Considering the measure for quality, in this replication the percentage of passed acceptance tests was not weighted by the difficulty score of the story in contrast with the original study. This was due to the impossibility of getting the information about the difficulty rank of each of the stories from the original study. This necessary change in the metric could result in relatively higher levels of quality compared to the original study.

Because of the changes in the metrics between the two studies, we had to leave out a direct comparison between the results of the original study and our replication, with respect to the overall level of external quality and productivity.

Table 4 Summary of the original and the replication study context variables

Variable	Original	Replication
Treatment applied	test-first v.s. test-last	test-first
Subjects (Recruited / Included)	35 / 24	53 / 30
Subject Unit	Individuals	individuals and pairs
Subject Type	Undergraduate students	Mix of undergraduate and graduate students
Programming Environment	Java, Eclipse, JUnit	Java, Eclipse, JUnit
Programming Task	Robert Martin's Bowling Scorekeeper	Robert Martin's Bowling Scorekeeper
Programming Task Type	Fine grained with incremental difficulty	Fine grained with incremental difficulty
Time to Complete the Task	Several sessions, working from home	One lab session $\sim$ 3 hours

4.4 Replication design

The subjects of this study were a mix of senior undergraduate and first year graduate students in information processing science at the University of Oulu, Finland. They took a 2-month graduate-level course about software quality and testing.[4] During the course, the students were given a theory lecture about TDD and a 3 hours lab hands-on tutorial about Java, Eclipse and JUnit followed by a simple exercise solved using TDD. The first lab session was followed by two additional sessions, in which the subjects were required to solve three, more complicated exercises using TDD.

During the last lab session, in which the experiment took place, 68 students (28 undergraduate and 40 graduate) were present. The students were informed about their participation in a study about TDD's effectiveness and they were asked to sign an informed consent form to allow the researchers to collect and use their software artifacts for the purpose of the experiment, although the experiment hypotheses were not disclosed. The subjects were also notified that their software artifacts would not be evaluated to determine their grades. Out of 68 students, 4 did not sign the form and therefore were excluded from the data collection. The subjects were asked to implement the same version of the Bowling Scorekeeper exercise used in the original study, with the difference that they had only three hours to complete the task. The exercise description was divided into 13 fine-grained stories of incremental difficulty. The number of implemented stories were later used to calculate quality and productivity metrics. Eleven subjects left the lab room without returning their artifacts. Since pair programming was encouraged (but not forced), the remaining 53 subjects self-organized into 10 individuals and 20 pairs.[5]

The subjects were provided with two web-based questionnaires to be answered anonymously. A pre-questionnaire to help assessing the subjects' skills and experience and a post-questionnaire to gather feedback after the conclusion of the study. The first questionnaire, completed by 60 subjects, showed that the majority had a good opinion about the effects of TDD and were will-

[4] `http://www.tol.oulu.fi/users/burak.turhan/815311a/`

[5] Three of the pairs included a third person, who declared that they were not confident with TDD and unable to complete the task, requesting to be a part of an existing pair in order to observe and learn pair programming practice.

ing to use this approach. In the pre-questionnaire, the subjects were asked to indicate how many programming courses they attended, and to rate their experience with the Java programming language. Figure 1 shows that all 60 subjects, but three, have taken at least one programming course before.[6] It also shows that the subjects have varying levels of previous experience with Java. Therefore, the results of the pre-questionnaire implies that there are no outstanding issues regarding subjects' skill levels regarding programming. In the post-questionnaire, completed by 53 subjects, 90% declared that they conformed to the TDD process through the whole session.

The tasks used during the course, including the experimental task, the pre and post questionnaires, and the test suite used to calculate the metrics are available for further replications.[7] Table 4 summarizes the context in which the replication took place.

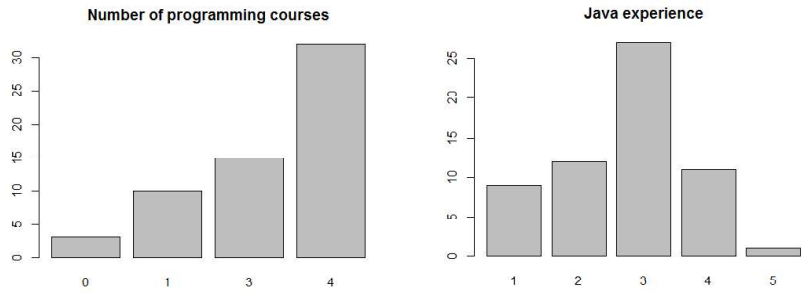

Fig. 1 Pre-questionnaire results about subject's experience.

4.5 Replication Data

The subjects of this study implemented 30 software artifacts as the outcome of the session. The artifacts were inspected by the authors to filter out irrelevant ones. An artifact was considered irrelevant if its Java classes, including the tests, were empty, or constituted only by stub methods. Particularly, the test classes tended to contain test cases with no assert statements that passed trivially. The assessment filtered out 14 software artifacts, while the remaining 16 were used to calculate the metrics used to test the hypotheses. Out of the final 16 artifacts 11 were developed by pairs and five were developed by individuals. The following metrics are collected from the final 16 artifacts, whose average size is 95 executable lines of code:

[6] Since Figure 1 data were collected anonymously, it includes drop-outs responses as well

[7] `http://cc.oulu.fi/~dfucci/lab_package2011.zip`

(#TESTS): Number of tests, measured by the count of the JUnit test cases. This is a ratio variable within the range $[0, \infty)$.
(QLTY): External quality, defined as the percentage of acceptance tests passed for the implemented stories.
(PROD): Productivity, measured as the percentage of implemented stories.
(SUBJ): Subject unit that implemented the software artifact. This is a nominal variable with two possible values: Individual (I) or Pair (P).

The dataset consisting of #TESTS, QLTY and PROD attributes was tested to discover outliers using both z-score and modified z-score methods [Cousineau and Chartier, 2010]. No significant outliers were found. Table 5 provides the raw data used in the analysis of this replication, as well as the descriptive statistics about the data.

Table 5 Dataset used in the replication

ID	#TESTS	QLTY	PROD	SUBJ
1	16	69	100	P
2	10	28	46	P
3	14	49	92	P
4	17	72	100	P
5	14	78	92	P
6	17	75	100	P
7	25	60	69	P
8	11	69	85	P
9	6	26	46	P
10	5	43	31	P
11	14	68	100	P
12	13	86	100	I
13	13	68	85	I
14	8	11	46	I
15	11	75	54	I
16	19	55	85	I
Min.	5.00	11.00	31.00	
1st Qu.	10.75	47.50	52.00	
Median	13.50	68.00	85.00	
Mean	13.31	58.25	76.94	
3rd Qu.	16.25	72.75	100.00	
Max.	25.00	86.00	100.00	

4.6 Replication Hypotheses

The number of tests (#TESTS) represents the independent variable in our analyses. The two dependent variables under investigation are external code quality (QLTY) and programmers productivity (PROD). Following the original study notation, the two hypotheses for this replication are formalized as 2Q and 2P as follows:

2Q - Higher number of tests implies higher external quality.
$H_{0-2Q} : QLTY = \beta_0 + \beta_1 \times \#TESTS, \beta_1 \leq 0$
$H_{A-2Q} : QLTY = \beta_0 + \beta_1 \times \#TESTS, \beta_1 > 0$

2P - Higher number of tests implies higher productivity.
$H_{0-2P} : PROD = \beta_0 + \beta_1 \times \#TESTS, \beta_1 \leq 0$
$H_{A-2P} : PROD = \beta_0 + \beta_1 \times \#TESTS, \beta_1 > 0$

In addition to the original study hypotheses, we formulate an additional hypothesis for this replication in order to control the confounding effects of having mix subject units (i.e. individuals and pairs). Please note that this hypothesis is tested for both QLTY and PROD as dependent variables.

2S - The subject unit (pairs and individuals) does not have an interaction with the number of tests in predicting the dependent variables.
H_{0-2Sa} : The slopes for the regression lines of each category in SUBJ are not significantly different
H_{A-2Sa} : The slopes for the regression lines of each category in SUBJ are significantly different
H_{0-2Sb} : The Y-intercepts for the regression lines of each category in SUBJ are not significantly different
H_{A-2Sb} : The Y-intercepts for the regression lines of each category in SUBJ are significantly different

We should point out that the original study hypothesis 2Q stated *"Higher number of tests implies higher external quality"* and 2P stated that *"Higher number of tests implies different external quality"* although the notation used for 2Q and 2P suggests that any significant β_1, either negative or positive ($\beta_1 \neq 0$), is plausible for rejecting the null hypotheses [Erdogmus et al., 2005]. Nevertheless, we preferred to formulate the alternative hypotheses consistent with their descriptions, hence $\beta_1 > 0$.

In order to evaluate 2Q and 2P, we use linear regression as in the original study. For the evaluation of 2S, we use analysis of covariance method. Please note that 2Sb hypothesis shall be assessed only if 2Sa fail to be rejected.

For assessing statistical significance, we use $\alpha = 0.05$ in all cases. All analyses are implemented with R statistical software v2.12.0, and package HH was used for the analysis of covariance (ANCOVA).[8,9]

4.7 Replication Results

In this section, we present the results of hypotheses testing. A summary is provided in Table 6.

4.7.1 Quality

The Spearman correlation between #TESTS and QLTY is 0.41, however it is not statistically significant (p-value = 0.11). Further, a significant relation

[8] `http://www.r-project.org/`
[9] `http://cran.r-project.org/web/packages/HH/HH.pdf`

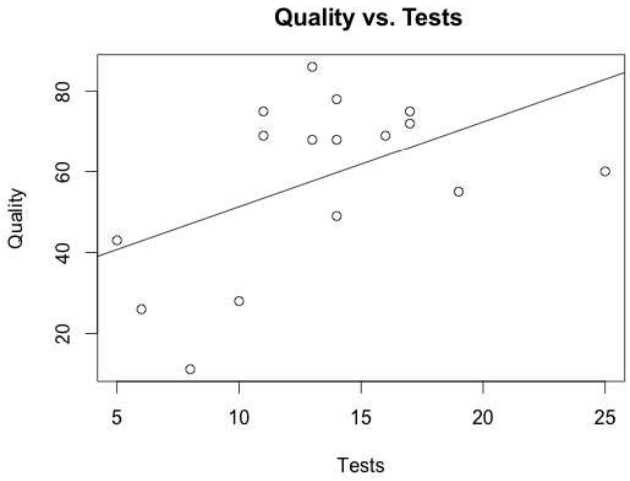

Fig. 2 QLTY as a function of #TESTS

between #TESTS and QLTY, as expressed in hypothesis 2Q, with a positive linear trend was not found. The linear regression between the two variables is expressed through the equation:

$$\text{QLTY} = 30.18 + 2.10 \times \#\text{TESTS} \tag{1}$$

Equation 1 is plotted in Figure 2. The significance test for linear regression coefficient, $\beta_1 = 2.10$ yields a p-value of 0.0513. Hence there is no statistically significant relationship between the number of tests and external quality. Therefore, we fail to reject the null hypothesis H_{0-2Q} .

4.7.2 Productivity

The Spearman correlation between the #TESTS and PROD variables is statistically significant with a value of 0.66 (p-value = 0.004). The relation between #TESTS and PROD variables, in the form of a positive linear relationship, is expressed by:

$$\text{PROD} = 37.30 + 2.97 \times \#\text{TESTS} \tag{2}$$

Equation 2 is plotted in Figure 3. The linear regression coefficient, $\beta_1 = 2.97$, is statistically significant with a positive value (p-value = 0.011). The coefficient of determination has a value of $R^2 = 0.37$, meaning that #TESTS accounts for 37% of variation in the data. This also indicates that other factors

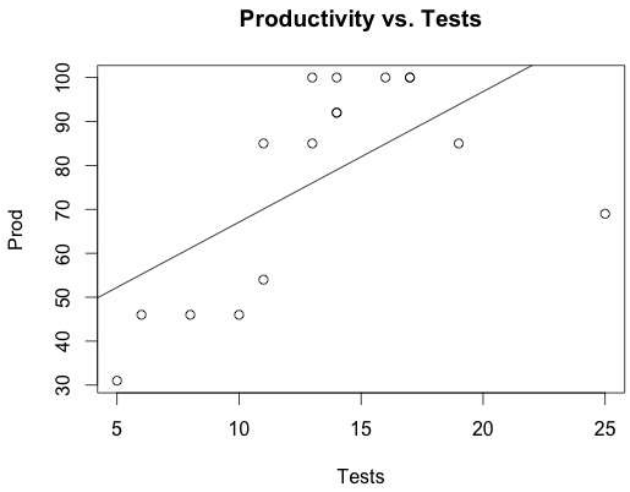

Fig. 3 PROD as a function of #TESTS. Note that two instances of datapoint (14,92) and (17,100) are plotted.

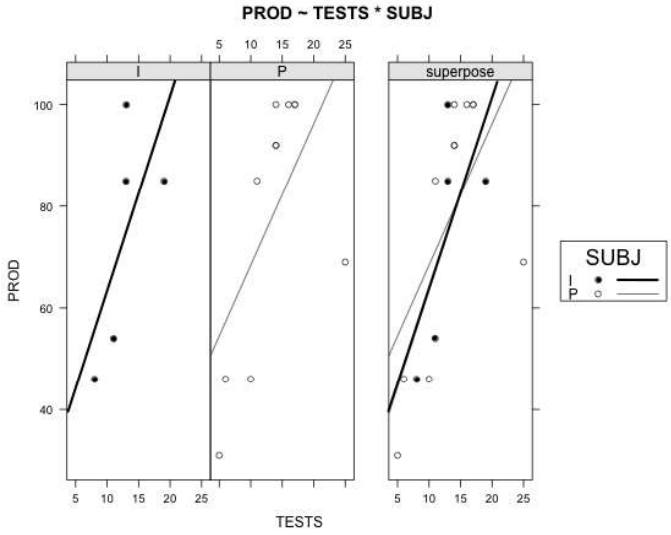

Fig. 4 Regression lines to test H_{0-2Sa} for $PROD$. Note that two instances of datapoint (14,92) and (17,100) are plotted.

are required to explain the remaining variability, which is inherently expected. Consequently, the null hypothesis H_{0-2P} can be rejected.

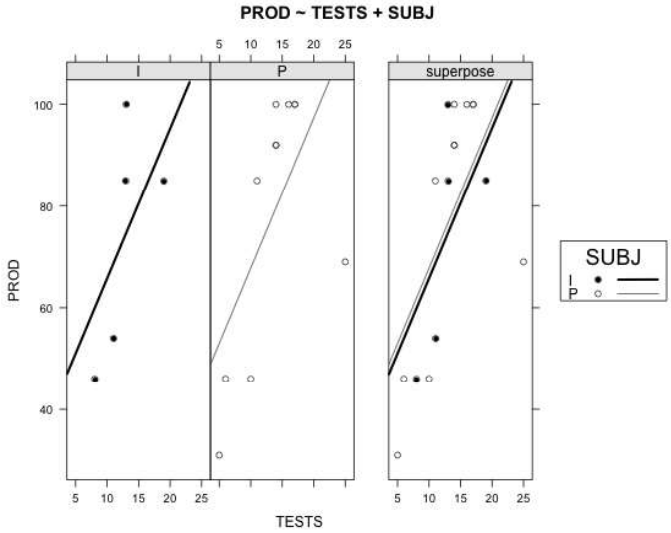

Fig. 5 Regression lines to test H_{0-2Sb} for $PROD$. Note that two instances of datapoint (14,92) and (17,100) are plotted.

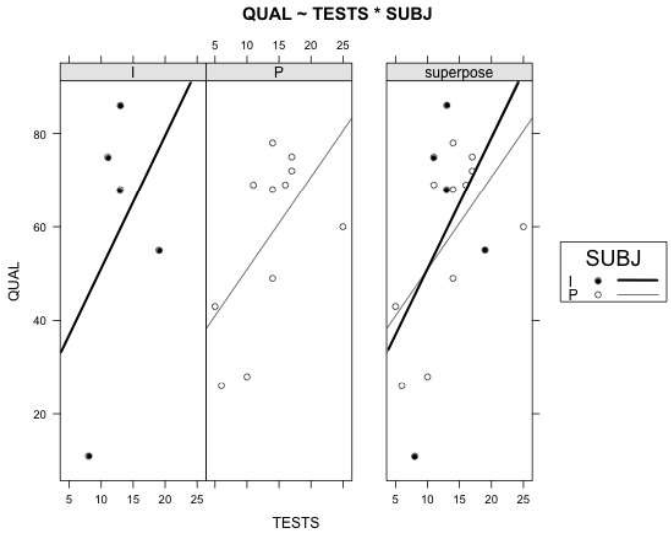

Fig. 6 Regression lines to test H_{0-2Sa} for $QLTY$

4.7.3 Interaction with subject unit

The regression analysis described in the previous sections did not take into account the impact of a second explanatory variable accompanying #TESTS. Possible interaction of the additional categorical variable SUBJ, which represents the subjects unit (individual or pair), with #TESTS is investigated in

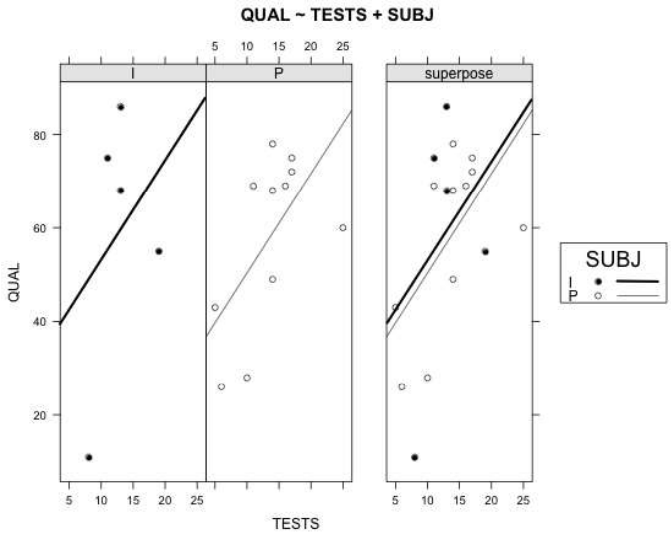

Fig. 7 Regression lines to test H_{0-2Sb} for $QLTY$

this section in order to check whether the results obtained so far depend on the subject unit.

In order to assess possible confounding effects of SUBJ, we carried out a sanity check analysis using the analysis of covariance (ANCOVA) method [Raubenheimer and Simpson, 1992]. The purpose of ANCOVA is to compare the two linear regression lines among the subgroups defined by SUBJ. Specifically, it checks whether the two subgroups' regression lines differ significantly in their slopes and Y-intercepts.

When SUBJ and the interaction term #TESTS*SUBJ were included in the regression model to predict the dependent variable QLTY, the regression coefficients for these variables were not statistically significant. In other words, the two different regression lines fit with the independent variable #TESTS and the grouping variable SUBJ, and do not have statistically different slopes (p-value=0.77). Therefore, H_{0-2Sa} failed to be rejected, and we proceeded with testing H_{0-2Sb}. Accordingly, we forced the two regression lines to be parallel (i.e. have a common slope). In this case we observed that Y-intercepts were not statistically different from each other (p-value=0.81), and H_{0-2Sb} failed to be rejected, as well. Hence, we found no evidence for interaction between #TESTS and SUBJ in predicting QLTY, and concluded that #TESTS is adequate to explain our data, regardless of the subject unit. Figure 6 shows the models used to test H_{0-2Sa}, whereas Figure 7 shows the models for testing H_{0-2Sb} .

Similarly, we checked the effects of inserting SUBJ and the interaction term #TESTS*SUBJ as additional predictors for PROD. The ANCOVA results revealed no significant difference between the slopes of the two regression lines (p-value = 0.73). Therefore, H_{0-2Sa} is failed to be rejected, and we proceeded

with testing H_{0-2Sb}. Again, we observed no significant difference between the Y-intercepts of the regression lines sharing a common slope (p-value = 0.85), and H_{0-2Sb} is failed to be rejected, as well. Therefore, there is no evidence for interaction between SUBJ and #TESTS in predicting PROD. Figures 4 and 5 shows the models used to test H_{0-2Sa} and H_{0-2Sb}, respectively.

Table 6 Summary results for replication hypothesis

Hypothesis name	Result
$2P$	Reject H_0
$2Q$	Failed to reject H_0
$2S$	Failed to reject H_0

5 Threats to Validity

In this section, we describe the limitations of this study in terms of threats to validity following the guidelines provided by Wohlin et al. [Wohlin, 2000].

5.1 Threats to construct validity

The necessary deviations from the original metrics used to measure the attributes under investigation introduces a threat to the construct validity for our replication. While this threat affects the measures of subjective concepts, i.e. productivity and external quality, it does not affect the number of tests since that attribute was measured directly with a simple count. This issue introduces a threat due to *mono-method bias* or *inadequate pre-operational explication of constructs*. On the other hand, number of tests reflect only one aspect of the construct and does not consider the quality of tests. While a finer-grained alternative could have been the assertion counts, we chose deliberately to stick with the original study's metric. Another threat to construct validity is the *interaction of different treatments*, specifically the confounding effects of the levels of subject unit in the replication study. However, we have handled this threat by constructing an additional hypothesis for the replication that checks the presence of interaction effects. There is a validity threat associated with *mono-operation bias* resulting from the use of the single task given to subjects as thoroughly discussed in the original study [Erdogmus et al., 2005]. Since this work is a replication study, we can not address this issue. Since the subjects were informed about their participation to the study, observed and encouraged to complete the tasks, and assisted for issues regarding the use of the IDE or tools, the Hawthorne effect [Bramel and Friend, 1981] could have occurred, affecting the subjects' normal behavior and introducing a threat due to *interaction of testing and treatment*. Though this could have affected subjects' productivity indirectly, we do not consider it a significant factor for our conclusion.

5.2 Threats to internal validity

Even though the majority of the subjects declared to have conformed to TDD, *process conformance* remains as a threat to internal validity for this replication. The subjects were informed about the importance of conformance and it was reinforced when possible. The artifacts were visually examined to check indicators that would suggest inadequate conformance (e.g. methods without test cases). Thus, we are quite confident that the subjects complied with TDD with a good degree of conformity even if this assertion cannot be supported by real data obtained through tools, e.g. [Johnson and Kou, 2007]. Though, initially we planned a formal control, we were not able to do so due to technical problems.

Our study inherits the bias introduced by the fine-grained nature of the task from the original study [Erdogmus et al., 2005]. Even though 59% of the subjects declared they would feel more confident in applying TDD to a fine-grained task, the overhead introduced by the nature of the task itself could affect the number of tests written. Eventually, the positive effects on short-term productivity would be visible at least for the most skilled subjects.

5.3 Threats to external validity

External validity could be affected by different factors. Apart from the small sample size and the simplicity of the task, the subjects were students with no or little experience in TDD before taking part in the course (only 8% declared they had used TDD before). As thoroughly discussed by the original authors [Erdogmus et al., 2005], the studies by Müller and Höfer, [Müller and Höfer, 2007] and Philipp [Philipp, 2009] compared undergraduate students with expert programmers showing that the effectiveness of TDD depends on the subjects skills, and only the most skilled students could perform closer to the average level of the experts, while most of the other students were way below it. Therefore, one limitation in our study lies in the lack of a formal assessment of skill difference between the original study and the replication groups of subjects.

Further, the high percentage of artifacts that were filtered out during the assessment (46%) represents another threat to the validity of the study, since it raises concerns regarding the subjects skills. However, the subjects skill levels regarding *programming* is not an issue as pointed out by the results of the pre-questionnaire. On the other hand, the main reason for the artifacts to be excluded from the analysis was due to *testing* related issues. This is inline with the lack of experience and comfort in using the JUnit tool, expressed by 67% of the subjects in the pre-questionnaire. We should also note that although the subjects have completed many programming courses before, this was the first software testing course they have taken. Therefore, we believe that the high percentage of irrelevant artifacts are due to the limited testing skills of the subjects rather than their programming skills.

6 Comparison and discussion of results

The results of this replication remains consistent with the results of the original study. Therefore, we build knowledge to support the original study findings. Specifically, we found that higher number of tests imply higher productivity, when TDD is applied. Ours is a specification of the original results which indicates that the relationship between tests and productivity exists independently from the development approach. A significant correlation between the number of tests and the external code quality was not found in this study, also confirming the results of the original study. All of these result remained consistent when checking for confounding effects of an additional explanatory variable, namely the subject unit.

This study has validated the result of the original study even though there were changes in the replication settings. Consequently, this replication has generated knowledge about the changes in context that are not influencing the study outcome. Further, the metrics for measuring quality and productivity were not exactly replicated, which improves our confidence in the hypotheses. In our results, the relationship between the number of tests and code external quality is open to question, the reason might lie in other context variables that are not yet evident after the analysis of both studies. On the other hand, the knowledge built by this study is that the number of tests is a good predictor for TDD programmer productivity, even when test-first component of TDD is not isolated, regardless of the subject unit (individuals or pairs), and in different time constraints, all of which strengthening the original study conclusions.

6.1 Relation between quality and productivity

The metrics for external quality and productivity used in this study appear to be intuitively related. Giving a fixed timeframe, the more the time dedicated to achieve higher quality the less the productivity; and vice versa, a developer could be more productive by implementing as much user stories as possible but neglecting quality. This intuition is supported by the data in this study as shown in Figure 8 (Pearson correlation coefficient=-0.74, p-value significant at 0.001). The subjects who achieved better productivity scores have lower quality scores. This means that most of the user stories delivered are incorrect or incomplete, although it is not possible from our data to say which ones. On the other hand, the subjects who performed better in terms of quality score were able to implement less than half of the user stories resulting in lower productivity scores. Previous studies mention such tradeoffs; based on empirical evidence, in the long run, quality seems to take higher precedence with the use of TDD [Marchenko et al., 2009,Sanchez et al., 2007].

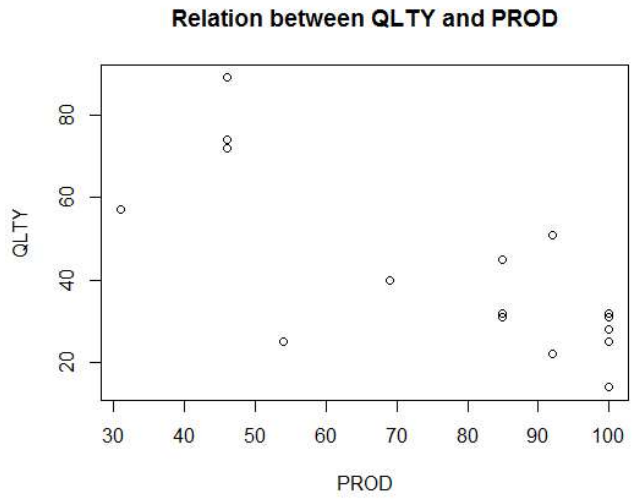

Fig. 8 Scatterplot for PROD and QLTY.

6.2 Interpretation of results

Looking at Figure 2 there is a high variation in terms of quality when the number of tests is low, for instance, with #TESTS<15 the QLTY level varies between 11 and 86, while with #TESTS>15 the variation of QLTY is between 55 and 75. A possible explanation is that, when the number of tests is low, the subjects' skill level plays an important role which could compensate for the low effort in testing. Such phenomena are noticed also by the original authors with whom we agree that, with an increasing numbers of tests, the level of quality tends to stabilize reducing the role played by the subjects' skill [Erdogmus et al., 2005]. If this is true, a consequence for the industry would be that a baseline level of quality can be reached by less skilled developers by pushing them to write more tests (i.e. by adopting TDD). Unfortunately, since in our study we could not relate the skill level, as measured by the pre-questionnaire, to the measure of quality, this conjecture cannot be supported by formal data analysis.

A similar variation is shown in Figure 3. With #TESTS<15, PROD varies between 31 and 100, while with #TESTS>15 the variation is between 85 and 100. We agree with the original authors that this might be due to difficulty encountered in learning TDD, but we are not able to state who, between the higher skilled or lower skilled subjects, would benefit the most in terms of productivity, due to the lack of data as pointed out earlier. If such conjecture was to be supported by data, one possible implication for industry would be that a homogeneous and high level of productivity can be reached by developers who apply TDD and are encouraged to write tests, once they passed the learning curve.

7 Conclusions

We replicated the Erdogmus et al. study about the role of the number of tests in TDD, and its relationship with external code quality and programmers' productivity. In the first stage of the original study, a high number of tests was associated with the use of TDD methodology. We aimed at validating/ refuting the original hypotheses that the numbers of tests is a predictor for external code quality and programmer's productivity as expressed in the second stage of the original study. Therefore, we conducted a partial, differentiated replication. In our replication some changes to the original context were necessary, we reduced the amount of time required for the subjects to complete the study's task, we enforced the use of TDD in all its parts (i.e. not isolating the test-first aspect), used different metrics to measure code quality and programmer productivity, and finally encouraged pair-programming among subjects. We also formulated an additional hypothesis to test the confounding effects of the subject unit (pair vs. solo).

We found no statistically valid correlation between tests and quality, confirming the results of the original study. These results were not dependent on the subject unit. This suggests that the relationship between the number of tests and external quality, if any, is hindered by other context variables. Also, as in the original study, we conclude that higher number of tests imply higher productivity. The results still hold after testing the effects of subject units. We have also observed that higher number of tests is coupled with higher baseline quality and productivity. Therefore, our recommendation for practice is to encourage developers towards implementing as many relevant tests as possible in order to achieve higher baselines for quality and productivity.

We reinforced the findings of the original study, at least when TDD methodology is used, and expanded the context in which the results still hold. Further, our confidence in the generalization of the results has increased, since some context variables were different than the original settings, namely the focus on the entire TDD process rather than isolating the test-first aspect and the different time constraints to complete the programming task.

The intentions for future work are threefold. We plan to carry out other replications considering this replication as the base study. Such replications will give us the chance to isolate the new variables we have discovered (e.g. the subjects' skill level for a baseline quality and productivity), to study their interactions and to clarify their impact. We are also planning to conduct a full replication of the original study with a test-last control group. Finally, in further replications we will also focus on the role of process conformance.

Acknowledgements This research is supported in part by the Finnish Funding Agency for Technology and Innovation (TEKES) under Cloud Software Program and the Academy of Finland with Grant Decision No. 260871. The authors would like to thank Hakan Erdogmus, Maurizio Morisio and Marco Torchiano for providing valuable insights along with the materials needed to conduct this replication. Authors also acknowledge the anonymous reviewers whose suggestions have significantly improved the earlier versions of the manuscript.

References

[Astels, 2003] Astels, D. (2003). *Test Driven development: A Practical Guide*. Prentice Hall Professional Technical Reference.

[Beck, 2003] Beck, K. (2003). *Test-driven Development: by Example*. The Addison-Wesley signature series. Addison-Wesley.

[Bhadauria, 2009] Bhadauria, V. (2009). *To Test Before Or To Test After-An Experimental Investigation Of The Impact Of Test Driven Development*. PhD thesis, The University of Texas at Arlington.

[Bramel and Friend, 1981] Bramel, D. and Friend, R. (1981). Hawthorne, the Myth of the Docile Worker, and Class Bias in Psychology. *American Psychologist*, 36(8):867.

[Brooks et al., 2008] Brooks, A., Roper, M., Wood, M., Daly, J., and Miller, J. (2008). Replication's role in software engineering. *Guide to advanced empirical software engineering*, pages 365–379.

[Carver, 2010] Carver, J. (2010). Towards reporting guidelines for experimental replications: A proposal. In *Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research*.

[Cousineau and Chartier, 2010] Cousineau, D. and Chartier, S. (2010). Outliers Detection and Treatment: a Review. *International Journal of Psychological Research*, 3(1):58–67.

[Dieste et al., 2010] Dieste, O., Fernandez, E., García, R., and Juristo, N. (2010). Hidden Evidence Behind Useless Replications. In *Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research*.

[Erdogmus et al., 2005] Erdogmus, H., Morisio, M., and Marco, T. (2005). On the Effectiveness of the Test-First Approach to Programming. *IEEE Transactions on Software Engineering*, 31(3):226–237.

[Flohr and Schneider, 2006] Flohr, T. and Schneider, T. (2006). Lessons learned from an xp experiment with students: Test-first needs more teachings. In Mnch, J. and Vierimaa, M., editors, *Product-Focused Software Process Improvement*, volume 4034 of *Lecture Notes in Computer Science*, pages 305–318. Springer Berlin / Heidelberg.

[George, 2002] George, B. (2002). Analysis and quantification of test driven development approach.

[George and Williams, 2003] George, B. and Williams, L. (2003). An initial investigation of test driven development in industry. In *Proceedings of the 2003 ACM symposium on Applied computing*, SAC '03, pages 1135–1139, New York, NY, USA. ACM.

[Huang, 2007] Huang, L. (2007). *Analysis and Quantification of Test First Programming*. PhD thesis, The University of Sheffield.

[Janzen and Saiedian, 2007] Janzen, D. and Saiedian, H. (2007). A leveled examination of test-driven development acceptance. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 719 –722.

[Johnson and Kou, 2007] Johnson, P. and Kou, H. (2007). Automated Recognition of Test-Driven Development with Zorro. In *AGILE 2007*, pages 15–25. IEEE.

[Juristo and Vegas, 2009] Juristo, N. and Vegas, S. (2009). Using differences among replications of software engineering experiments to gain knowledge. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, pages 356–366, Washington, DC, USA. IEEE Computer Society.

[Keefe et al., 2006] Keefe, K., Sheard, J., and Dick, M. (2006). Adopting xp practices for teaching object oriented programming. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, pages 91–100, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

[Madeyski, 2010] Madeyski, L. (2010). *Test-driven development: An empirical evaluation of agile practice*. Springer-Verlag New York Inc.

[Marchenko et al., 2009] Marchenko, A., Abrahamsson, P., and Ihme, T. (2009). Long-term effects of test-driven development a case study. In Abrahamsson, P., Marchesi, M., and Maurer, F., editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 31 of *Lecture Notes in Business Information Processing*, pages 13–22. Springer Berlin Heidelberg.

[Melnik and Maurer, 2005] Melnik, G. and Maurer, F. (2005). A cross-program investigation of students' perceptions of agile methods. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 481 – 488.

[Müller and Höfer, 2007] Müller, M. and Höfer, A. (2007). The Effect of Experience on the Test-driven Development Process. *Empirical Software Engineering*, 12(6):593–615.

[Pančur and Ciglarič, 2011] Pančur, M. and Ciglarič, M. (2011). Impact of test-driven development on productivity, code and tests: A controlled experiment. *INFORMATION AND SOFTWARE TECHNOLOGY*, 53(6):557–573.

[Pedroso et al., 2010] Pedroso, B., Jacobi, R., and Pimenta, M. (2010). TDD Effects: Are We Measuring the Right Things? *Agile Processes in Software Engineering and Extreme Programming*, pages 393–394.

[Philipp, 2009] Philipp, M. (2009). Comparison Of The Test-Driven Development Processes Of Novice And Expert Programmer Pairs.

[Rafique and Misic, 2012] Rafique, Y. and Misic, V. B. (2012). The effects of test-driven development on external quality and productivity: A meta-analysis. *IEEE Transactions on Software Engineering*, 99(PrePrints).

[Raubenheimer and Simpson, 1992] Raubenheimer, D. and Simpson, S. (1992). Analysis of Covariance: an Alternative to Nutritional Indices. *Entomologia experimentalis et applicata*, 62(3):221–231.

[Sanchez et al., 2007] Sanchez, J. C., Williams, L., and Maximilien, E. M. (2007). On the sustained use of a test-driven development practice at ibm. In *Proceedings of the AGILE 2007*, AGILE '07, pages 5–14, Washington, DC, USA. IEEE Computer Society.

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

[Turhan et al., 2010] Turhan, B., Layman, L., Diep, M., Erdogmus, H., and Shull, F. (2010). *How Effective Is Test Driven Development?* O'Reilly Media.

[Wohlin, 2000] Wohlin, C. (2000). *Experimentation in Software Engineering: an Introduction*, volume 6. Springer.

[Xu S, 2009] Xu S, L. T. (2009). Evaluation of test-driven development: An academic case study. *Studies in Computational Intelligence*, 253:229–238.

[Yenduri and Perkins, 2006] Yenduri, S. and Perkins, L. (2006). Impact of using test-driven development: A case study. *Software Engineering Research and Practice*, pages 126–129.