Towards an Operationalization of Test-driven Development Skills: An Industrial Empirical Study

Article *in* Information and Software Technology · September 2015

DOI: 10.1016/j.infsof.2015.08.004

6 authors, including:

Davide Fucci
University of Hamburg

Natalia Juristo
Universidad Politécnica de Madrid

Óscar Dieste Tubío
Universidad Politécnica de Madrid

Markku Oivo
University of Oulu

Towards an Operationalization of Test-driven Development Skills: An Industrial Empirical Study

Davide Fucci[a], Burak Turhan[a], Natalia Juristo[a,b], Oscar Dieste[a], Ayse Tosun Misirli[c], Markku Oivo[a]

[a] *M-Group, University of Oulu*
[b] *Escuela Tecnica Superior de Ingenieros Informaticos, Universidad Politecnica de Madrid*
[c] *Faculty of Computer and Informatics, Istanbul Technical University*

Abstract

Context: The majority of the empirical studies on Test-driven development (TDD) are concerned with verifying or refuting the effectiveness of the technique over a traditional approach, and they tend to neglect whether the subjects possess the necessary skills to apply TDD, though they argue such skills are necessary.
Objective: We evaluate a set of minimal, a priori and in process skills necessary to apply TDD. We determine whether variations in external quality (i.e., number of defects) and productivity (i.e., number of features implemented) can be associated with different clusters of the TDD skills' set.
Method: We executed a quasi-experiment involving 30 practitioners from industry. We first grouped the participants according to their TDD skills' set (consisting of a priori experience on programming and testing as well as in-process TDD conformance) into three levels (Low-Medium-High) using k-means clustering. We then applied ANOVA to compare the clusters in terms of external quality and productivity, and conducted post-hoc pairwise analysis.
Results: We did not observe a statistically significant difference between the clusters either for external software quality ($\boldsymbol{F(2, 27 = 1.44}$, $\boldsymbol{p = .260}$), or productivity ($\boldsymbol{F(2, 27) = 3.02}$, $\boldsymbol{p = .065}$). However, the analysis of the effect sizes and their confidence intervals shows that the TDD skills' set is a factor that could account for up to 28% of the external quality, and 38% for productivity.
Conclusion: We have reason to conclude that focusing on the improvement of

TDD skills' set investigated in this study could benefit software developers in improving their baseline productivity and the external quality of the code they produce. However, replications are needed to overcome the issues related with the statistical power of this study. We suggest practical insights for future work to investigate the phenomenon further.

Keywords: Test-driven development, process conformance, software quality, developers' productivity.

---

1. Introduction

Test-driven development (TDD) is a software development technique in which the development is guided by writing unit tests. It was popularized in the late 1990s as part of Extreme Programming [1]. A developer using TDD follows four steps:

1. Write a unit test for the functionality she wants to add.
2. Run the unit test to make sure it fails.
3. Write only enough production code to make the test to pass.
4. Refactor both production and test code, and re-run the tests.

TDD is claimed to yield better results than traditional approaches to software development (e.g., when unit tests are written after the intended functionality is considered *completed* by the development team) in terms of developers' productivity, external quality (e.g., reduced number of defects), maintainability, and extensibility [2, 3]. However, empirical investigations of the effects of TDD are contrasting [4, 5], arguing that the results are influenced by several variables (e.g., academic vs. industrial settings), including the *skills of developers*.

Literature reviews on TDD conclude that the application of the technique—and subsequently the manifestation of its postulated benefits—requires some skills [5, 6]; however, these studies do not indicate what these skills are. We started our investigation on skills with students in a previous study [7]. In that context, we looked at their pre-existing knowledge regarding two practical skills: proficiency with programming language and unit testing (UT). When the subjects tackled a small programming task using TDD, we found that such skills had little impact on their productivity— defined as the output (e.g., parts of the task completed) per unit of effort (e.g., time to complete the task). No significant relationship was observed regarding the quality of the software they produced—e.g., the defects found in the parts of the task which were completed by the subjects. In the same study, we acknowledged that other skills must be present in order for TDD developers to achieve the benefits advocated by TDD supporters.

With these motivations based on existing literature and our previous work, we incorporate in this study another practical skill, which we call *TDD process conformance*, along with programming language and unit test skills. *TDD process conformance* represents the ability of a developer to follow the TDD cycle. Together, these three skills represent our *TDD skill set*. Further, we used a more realistic task to overcome the limitations of small programming tasks, and recruited professional developers for the study. Consequently, the research goal of this work is the following:

Understanding the effect of the developers' TDD skills on external quality and productivity

In our previous studies [7, 8, 9] we have investigated the role that each skill plays *individually* with student subjects working on toy tasks. We now focus on the impact the skills have, when taken *together*, on the outcomes of interest, by performing a quasi-experiment involving 43 professional software developers (30 after mortality) without prior working experience in TDD. The developers were trained during a week-long workshop and then asked to implement new features of a legacy system using TDD. Finally, we evaluated the composite effect of their skills on their performance in terms of external quality and productivity. Hence, we contribute to the existing knowledge by:

- Empirically investigating an anecdotal claim: that is, TDD requires skills to manifest benefits, with professional developers.
- Building a model for quality and productivity that takes into account a set of practical skills (section 3)
- Providing initial empirical evidence that further investigation of the proposed *TDD skill set* are worth pursuing (section 5)

The strong points of our study lie in the settings (section 4) in which it was conducted. In particular, we:

- Analyse data collected from professional software developers.
- Utilize a near real-world, brown-field task, rather than a toy, green-field, task (see Section 4.2 and Appendix B).
- Quantify process conformance analytically, rather than relying on self reports.

The rest of the paper is organized as follows. In Section 2 we present the existing literature related to our research, in Section 3 we define the TDD skill set used in our study. Section 4 explains the details of our empirical study design. Sections 5 and 6 reports the results and associated discussions. We address the threats to the validity of our study in Section 7. We conclude the paper in Section 8.

2. Related Work

Test-driven development has been the subject of several secondary studies. The systematic literature review by Turhan et al. [5]—covering 32 empirical studies—found positive effects on external quality, whereas the productivity results were inconclusive, when TDD was used across different settings. The meta-analysis by Rafique and Misic [4] is of interest when looking at how experience works with the postulated TDD effects. The work covers 10 years of TDD publications, from 2000 to 2011, in 25 selected primary studies. The authors focused part of their analysis on comparing studies whose subjects had different kinds of experience, i.e., academic vs. industrial. The results show improvement for professionals in terms of external quality, but a deterioration of productivity compared to student subjects.

In a recent systematic literature review, Munir et al. [10] classified the primary studies according to relevance and rigor. In particular, relevant studies, i.e., studies dealing with realistic settings that have applicability in an industrial context, show that TDD benefits professional developers in terms of external quality at the expense of productivity. Nevertheless, the authors suggest that there is a lack of industry experiments dealing with real-world systems and long-term studies.

Based on the big picture provided by the systematic literature reviews, it appears that the goal of TDD research (including the secondary studies) is to gather evidence about TDD beneficial effects over a traditional approach to software development, like test-last development. We acknowledge the importance of such research effort, but we also note that the majority of the empirical work pays insufficient attention to whether the subjects possess the necessary skills, and apply such skills in a test-driven fashion. Moreover, prior research defines experience in terms of subject roles, e.g., students vs. professionals.

Latorre [11] studied the effects of the application of TDD by a pool of professional software developers (i.e., having skills with Java programming, and unit testing in JUnit but not TDD) to a real-world, although simple, software system, over a one-month period. The author shows that the developers were able to apply TDD correctly after a short practice and retain such knowledge later in their daily work. When the subjects were considered according to their seniority (i.e., junior, intermediate, and senior), the results show that the ability to readily apply TDD initially depends on experience. In fact, senior developers were able to achieve a high level of conformance to the process after few iterations, while intermediate and juniors needed more time, after which, all the subjects reached a plateau level between 80% and 90%. On the other hand, experience had an impact on productivity. Only the most expert subjects were able to keep the productivity at the level of a traditional development approach (the initial part of the system was developed without employing TDD), while the less experienced ones lagged behind due to the problems they encountered with refactoring and design decisions. Nonetheless, all the subjects delivered a correct and functioning version of the system. Therefore—although not explicitly mentioned by the author—external quality does not seem to be affected by the subjects' experience or level of conformance. The author advises that similar studies should be repeated by taking into account different levels of experience with the programming language, unit testing, and tools, as well as real-world application, since such factors might affect the adoption of TDD.

Another study inspecting the role of experience and process conformance in TDD settings is the controlled experiment by Müller and Höfer [12], in which experienced and novice developers were compared. The experts in this case also had previous knowledge of Java (average 6.4 years), JUnit (average 4.3 years), and TDD (average 3.4 years); whereas, the novices were Master's students participating in an Extreme Programming course. The results show that experts are able to achieve better productivity (time to complete the task) but not quality (passing acceptance tests) for which a non-significant difference was found. Nevertheless, the authors conjecture that the observed difference might be due to the novice subjects' general lack of programming experience. Process conformance was measured, but as a separate factor from the developers' experience. The authors report that the experienced subjects adhered more to the process than novices, by a significant amount.

3. A Skill Set for TDD

Our goal in this paper is to make a holistic analysis of the skills rather than focusing on them individually. Therefore, we include three skills, i.e., programming and testing skills as well as *TDD process conformance*, to define a *TDD skill set.*

Although existing literature acknowledges that skills matter when applying TDD, none indicates the necessary ones. For example, Causevic et al. identified the lack of developers' skills as one of the main impediments to the adoption of TDD by industry, though they do not indicate specifics about these skills [6].

We have previously investigated software development related skills, such as programming and testing-related skills with students [7], and showed that they marginally impact only the subjects' productivity. We argue that the positive effects on external quality and developers' productivity [1] should be visible once a series of TDD cycles *actually* take place. Based on our experience in teaching TDD in the university courses as well as running TDD workshops at companies [13], we also argue that a test-driven development endeavor comprises several cognitive efforts: the ability to slice a requirement in a simple enough task, lay down such task in the form of a unit test, make the test pass by writing the minimal code necessary, identify refactoring opportunities, and perform the right refactoring. Hence, we consider the ability of a developer to follow TDD as a necessary skill, and we call this ability *TDD process conformance*. Systematic literature reviews also highlight *TDD process conformance* as a potential factor to explain the different results reported in primary studies [5].

We previously studied process conformance in isolation as well, and we did not observe any impact on external quality or productivity when students were used as subjects [9]. As Lattore reports, developers with professional experience are able to quickly pick-up the technique (e.g., TDD) and conform to it after a short period of practice [11]. Nevetheless, from our experience we observed that students tend to perform at the same level as professionals (at least in terms of internal code quality) when TDD is newly introduced to them [14], hence we have grounds to reflect our previous experience with students to the professional subjects of this study.

For programming and testing skills, we are particularly interested in gauging the subjects' experience with unit testing related abilities like testing patterns and good practices, as well as the subjects' experience with a particular programming language. Java was selected as the study's *lingua franca* due to the diversity of programming languages used regularly by the subjects. In fact, not all the subjects use Java in their daily professional activities. Even though we could have allowed them to choose the language freely, this would have made the results difficult to compare because of the confounding factor of different programming languages and available tools to support development. However, the results from a survey among 13,000 professional developers by Meyerovich and Rabkin show that a professional developer has a 94% chance of knowing an object-oriented programming language; the percentage increases to 97% if the subject holds a degree in computer science [15]. In this study, 21 subjects out of 30 considered have at least a B.Sc. in computer science, two do not hold a degree, and the rest have a mix of other science and engineering degrees. Moreover, the training was carried out using Java as a reference language due to the availability of tools, such as the one used to measure process conformance.

Since TDD is a development activity that leverages unit tests, unit testing skills can have a significant impact on the application of TDD itself. After all, tests are the steering factor for TDD style implementation. We selected unit testing, rather than a more specific skill like familiarity with JUnit, since the concepts behind it can be easily implemented in the framework of reference for the selected programming language. At the beginning of the workshop, we gave the subjects a tutorial on the Java standard testing framework, JUnit, which was used throughout the rest of the workshop with references to how testing principles can be used with these tools.

In summary, the operationalisation of what we call *TDD skill set*, which we use as a criterion to cluster the subjects and compare their performances, is constituted by the following:

- TDD Process conformance (CONF)
- Developers' knowledge of Java programming (JAVA)
- Developers' knowledge of unit testing (UT)

4. Study Definition

An overview of the study is presented in Figure 1. The study seeks the answers to the research questions presented in Section 4.1. We recruited subjects from two companies, in the context of a workshop about unit-testing and TDD (Section 4.2). We assessed the subjects' skills in Java development and unit-testing at the beginning of the workshop. During the workshop the subjects carried out a brown-field, real-world task (Section 4.3). Subsequently, we collected the necessary data to extract TDD process conformance—which forms, together with the data about the development and unit-testing skills, the TDD skills set—and the two outcomes of interests, external quality and productivity (Section 4.4). We then analyzed the data, using the methods reported in Section 4.5, in order to test the hypotheses (Section 4.6) associated with the research questions.

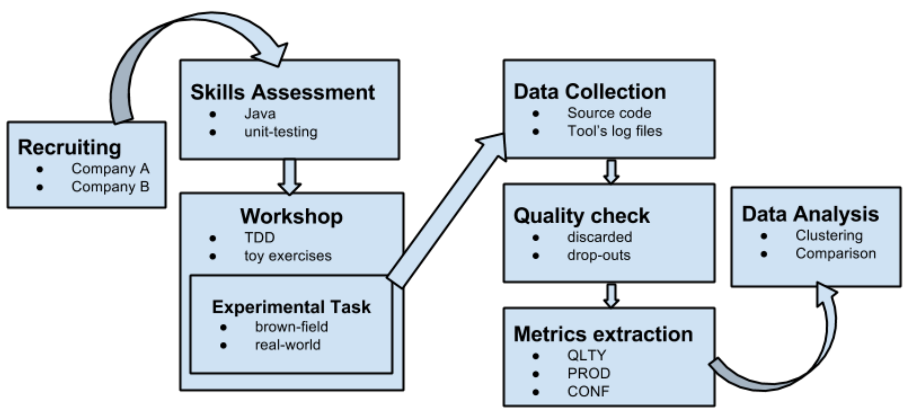


Figure 1: Overview of the study's main aspects.

4.1. Research Questions

In this paper, we focus on the effects of TDD on external quality of a software system (e.g., the extent to which the system adheres to given functional requirements, specifically in the form of acceptance tests), and productivity (e.g., the capacity of a developers to complete the features of a given system), from the perspective of the TDD skill set explained earlier. This is in line with our previous studies [9, 16]. Hence, the research question is divided into two separate questions:

RQ_{QLTY} — Does the external quality of a software system depend on the TDD skill set of its developers?

RQ_{PROD} — Does the productivity of developers depend on their TDD skill set?

The operationalization of the constructs under study, external quality, productivity, and the three components of the TDD skill set is described in Section 4.4.

We employ a quasi-experiment design based on the nature of the constructs we are dealing with. In other words the possibility of executing a controlled experiment was discarded since it is not possible, for example, to assign subjects to a skill level, very least do it randomly [17].

4.2. Context & Subjects

Table 1: Subject recruitment breakdown.

	Company				
Status	A (site 1)	A (site 2)	A (site 3)	B	Total (by status)
Recruited	7	11	6	19	43
- Drop-outs	0	0	2	0	2
- Discarded	2	5	1	3	-11
Total (by company)	5	6	3	16	30

This study was conducted during five-day long TDD workshops held in two different companies at four different sites (three at Company A and one at Company B). The companies operate in different domains: Company A produces security-related solutions, whereas Company B operates in the entertainment and gaming field. In particular, 24 professional software developers participated in the workshop from Company A and 19 from Company B. Table 1 shows the breakdown of the subjects recruited/included for the study. According to the pre-questionnaire administered to the recruited subjects, none had used TDD in their daily work before, although eight stated that they had participated in a TDD training or workshop in the past. Two of the recruited subjects (both in Company A, site 3) could not attend the study session due to personal issues.

Although data was collected over different sites, we do not consider this study par to the *multi-center trials* typical of health sciences. Multicenter trials, for example, are required in later stages of a drug approval process, and address side effects and safety risks [18]. We are not interested in the geographical or environmental factors, nor in comparing different sites. However, using data collected from several sites improves the power of the analysis and the external validity of the results [19].

On the first day, before the workshop started, the subjects filled-up a pre-questionnaire to self-assess their skills. Two of the items on the questionnaire were then used to operationalize Java programming and unit-testing skills metrics.

The workshop was organized as a coding dojo. A coding dojo is a place dedicated to the deliberate practice of programming activities like unit-testing, test-driven development, refactoring, pair-programming. In a coding dojo the participants are not focused on the results of their development activity, but on learning-by-doing and assimilating the practice [20, 21]. Each day the participants practiced unit testing and TDD using coding katas. A coding kata is a simple exercise that allows a programmer to focus on the skills she wants to practice, i.e., TDD, without being overwhelmed by the complexity of the task. These exercise sessions, both individual and in groups, were interleaved with a discussion on topical points led by the instructor (one of the authors of this paper).

The coding dojo activities and the use of katas were used only to train the subjects during the workshops. One example of the katas used during the

Table 2: Study main context variables

Variable	Value
Development technique	Test-driven development
Subject	Professional developers
Programming language	Java 6
Testing framework	JUnit 4
IDE	Eclipse 3.6
Task type	brown-field (1033 SLOC, 17 classes)
Duration	4 hours
Place	Company's site

workshop is reported in Appendix A.

The experiment was carried out on a task of near real-world complexity rather than a simple programming kata. The development environment was provided through a virtual machine, installed on the subjects' machine, which included Windows 7 as operating system, Java version 6, JUnit version 4 and Eclipse version 3.6 as well as an Eclipse plugin used to measure the *TDD process conformance*. Table 2 summarizes the context variables in which the study took place.

The plugin tool, Besouro [22], captures and classifies low-level development events into episodes that are labelled as TDD compliant or not according to a set of heuristics [23, 24]. Kou et al. validated these heuristics by comparing the manual evaluation of the activity of 28 developers (18 in academia and 10 in industry) against the automated evaluation performed following the heuristics. The results demonstrated an accuracy of TDD episode recognition between 85-90% [23]. The tool, Besouro, was evaluated by 14 subjects from the Brazilian Agile community [22]. Although other tools—capable of quantifying TDD process conformance—exists [23, 24, 25], we decided to use Besouro, since the tool leverages *empirically validated* heuristics and does not require external dependencies (e.g., external server, source code management system) other than the

IDE; therefore it is easy to deploy it at the companies' sites. Finally, due to the availability of such tools, we ruled out measuring process conformance via subjects' self assessment, following the recommendation of literature reviews on the topic [5]. During the last session, lasting four hours, the subjects tackled the experimental task. We gathered data from this session, using Besouro, in order to answer our research questions.

4.3. Objects

During the last workshop session—when the actual study took place—the participants individually tackled a near real-world complexity task. The task used was brown-field, similar to the majority of projects the participants usually face during their day-to-day jobs. The task involved adding features to a three-tier architecture (graphical user interface layer, business logic layer, and data access layer) system. In particular, the subjects were asked to implement the new functionalities in the business logic layer, since the data access and UI layers were already in place. The system does not belong to any of the domains in which the subject currently work in their day-to-day activities.

All the subjects were required to implement three functionalities, corresponding to three sub-tasks, of incremental difficulty within the system; nevertheless, they were free to choose the order in which to tackle them (see Appendix B). The first sub-task is more algorithmic, as it requires the subjects to implement a non-trivial formula. The other two are more architecture-oriented since they require the subject to interact with some of the existing classes in the system.

The existing system provided to the participants contains 13 Java classes and four interfaces (1033 LOC). The business logic layer, in which the developers are supposed to implement the required functionalities, included three existing classes (92 LOC). The system was accompanied by one smoke test (6 JUnit assertions, 38 LOC), i.e., an high-level test that vertically exercises the existing components and provides an example of the API used by such components to communicate between them. Other than for the smoke test, no other existing tests are given to the subjects in order to simulate a common real-world scenario. No particular domain knowledge was required to understand the system as we also provided its documentation, including a class diagram and a textual description of the Java classes (see Appendix C).

By design, the task was difficult, but not impossible, to complete in the allotted time. In order to test this requirement, the task was implemented by members of our research team, professional developers and students in our courses. During these trials we observed that some people (including students) were able to finish it in time, whereas the majority could not. This has the positive effect of adding realism to the settings due to the time pressure.

The complexity of the task is confirmed by the answers the participants gave to the post-questionnaire. None of the subjects thought the task was *Easy*; the majority (53%) assessed it as *Somewhat difficult* and 20% found it *Difficult*. Nevertheless, the subjects did not seem discouraged since most of them (57%) indicated the task was *Enjoyable* and only two of them found it *Boring*.

The participating subjects were asked to complete as many of the required functionalities in the time allotted for the session, using TDD as their development methodology. At the end of the workshop, we collected the subjects' virtual machine containing the solutions to the task. Out of the 43 solutions collected, 11 were discarded after the following quality checks:

- Remove the solutions for which no data about process conformance was collected: This was due to a malfunctioning of the tool, i.e., the tool did not save correctly on disk the file containing the necessary information to measure process conformance. We suspect that this is due to a particular configuration, heap allocated on memory, of the Java Virtual Machine on the subject computer. We do not exclude that another cause of the problem could lie in the subjects' machine not meeting the recommended hardware requirements to run the virtual machine, though it was communicated to them before the session. Eight solutions were discarded.

- Remove the solutions that cannot be compiled: This was done because an acceptance test suite need to be run against the code provided by the subjects to measure external quality and productivity. Three solutions were discarded.

Finally, two subjects (both in Company A, site 3) could not attend the study session due to personal issues as indicated earlier (see Table 1). Therefore, a total of 30 observations were used for the data analysis.

4.4. Data Collection and Metrics

TDD Skill Set: For experience related *TDD skill set* components, we used a questionnaire—administered before the workshop—to measure the subjects' self-perceived Java programming (JAVA) and unit testing (UT) abilities. Such self-perceived metrics are considered reliable for measuring programming experience [26]. We used a four-point Likert scale, i.e., without a midpoint, in order to force the subjects to make a choice and avoid neutral answers. The questionnaire statements are:

JAVA — "Rate your skill with the Java programming language"

UT — "Rate your skill with unit testing"

The answer could be selected from *None*, *Novice*, *Intermediate*, and *Expert*. Hence, both JAVA and UT were measured through ordinal scales with four levels mapped onto numerical values from 0 (*None*) to 3 (*Expert*). Formally, $JAVA, UT \in \{0, 1, 2, 3\}$.

The third component of our *TDD skill set* model is the ability to follow the TDD process, measured by the variable **CONF**. The log files created by the tool installed as a plugin in the development environments were collected to calculate this component of *TDD skill set*. The value of CONF corresponds to the ratio of TDD-compliant episodes to the total number of development episodes identified by the tool, normalised by 100 ($CONF \in [0, 100]$), as presented in Equation 1. For example, an episode represented in the following sequence of events logged by the IDE is categorized as TDD compliant:

```
Create TestFoo.java
Edit TestFoo.java ADD test() METHOD
RunTest TestFoo FAIL
Create Foo.java
Edit Foo.java ADD bar() METHOD
RunTest TestFoo OK
Edit Foo.java CHANGE bar() METHOD
RunTest TestFoo OK
```

Notice that, in the example, the last two steps—indicating a refactoring activity—are optional.

$$CONF = \frac{\#episode(TDD)}{\#episode(TDD) + \#episode(\neg TDD)} \times 100 \tag{1}$$

The measure of JAVA, UT, and CONF are used to establish *TDD skill set* (and the associated levels), which is the only explanatory (or independent) variable of the study.

External quality and productivity: The source code, developed by the subjects, was collected and used for measuring the two outcomes: external quality and productivity. The three new functionalities required by the task were broken down into 11 user stories, hidden from the subjects. We used an acceptance test suite, having a test class associated with each of the user-stories, to measure external quality and productivity. The details of each test class, including which of the experimental task's feature is targeted, are provided in Table 3.

The acceptance test suite was developed as follows: We already had an initial acceptance test suite developed alongside with the task. For the purposes of this study, one team member broke down the sub-tasks into user stories and this was verified by two other members. Existing acceptance tests were mapped to the user stories, and then new tests were added when necessary by one of the members who verified the user stories, following the equivalence partitioning testing strategy to identify boundary cases. The final suite is then verified by

Table 3: Summary of the acceptance tests suite used to calculate $QLTY$ and $PROD$.

Sub-task	JUnit test class	# JUnit tests cases	# JUnit asserts
1	US1	4	4
	US2	3	12
	US3	3	12
	US4	4	4
2	US5	10	26
	US6	8	12
	US7	7	17
3	US8	1	1
	US9	2	11
	US10	2	13
	US11	4	20
	Total	48	132

the team member who worked on breaking down the subtasks into user stories. Further, the acceptance test suite was verified against our own implementation, and also used in the context of a course to evaluate student projects, and it was observed, in our experience, to be able to differentiate among different levels of quality.

The metric used to gauge external quality (QLTY), is calculated on a user-story basis. We introduced the concept of *tackled user story* (TUS) to identify which user stories were actually engaged by the subjects. A particular user-story is considered *tackled* if at least one of the acceptance tests associated with it is passing. The following formula is used to calculate QLTY:

$$QLTY = \frac{\sum_{i=1}^{\#TUS} QLTY_i}{\#TUS} \times 100 \tag{2}$$

Equation 2 represents $QLTY$ as the sum of the quality of each tackled user-story divided by the number of tackled user stories, normalised by 100. $QLTY_i$ is the quality of the i-th user story that constitute the task, calculated as follows:

$$QLTY_i = \frac{PASS(assert_{(i)})}{TOTAL(assert_{(i)})} \tag{3}$$

In other words, $QLTY_i$ is the ratio of passing assert statements in the acceptance tests associated with the *i-th* user story. We adopt this metric because we are interested in the quality of the *delivered* features. Therefore, if some features are not worked on, their inclusion in the calculation would have unfairly affected the metric. As a tradeoff, the $PROD$ of such cases will be lower, as will be discussed next. Following Equations 2 and 3, $QLTY \in [0, 100]$.

In general, productivity is thought to be the amount of work done during a certain amount of time. Nonetheless, the time to implement the task was fixed and none of the subjects completed the task before the end of the study. The time capping was chosen so that the task could be difficult, but not impossible to complete. This allow us to add to the settings the effect of *time-pressure* that developers usually experience when a deadline is approaching. Hence, $PROD$ is not measured in terms of time, (e.g., the less time used, the better), but rather on the parts of the system a subject is able to complete [27].

We considered counting the number of user stories completed by each subject as a productivity measure, using the test cases as a criteria to define when a user story is complete. The setback of this approach is that having only 11 user-stories might not be enough to differentiate between the subjects. Hence, we followed a more granular approach and calculated PROD as the percentage of passing asserts (regardless of the user-story they are associated with) in the acceptance test suite.

$$PROD = \frac{PASS(assert)}{TOTAL(assert)} \times 100 \tag{4}$$

Following Equation 4, $PROD \in [0, 100]$. Therefore, $PROD$ is a representation of how much of the required new functionalities has been developed. $QLTY$ and $PROD$ are the dependent variables for this study.

4.5. Methods

In order to check the impact of skills on the two outcomes, we divided the subjects into groups *a posteriori* since *a priori* random assignment, as assumed for controlled experiments, was not possible (i.e., it is not possible to know the *TDD process conformance* for a subject before she completes the experimental task). We used a typical clustering algorithm to form the non-experimental groups [1]. [17]. The study is carried out in two phases:

1. Clustering the subjects according to their *TDD skill set*.
2. Comparing clusters (groups) of subjects with different *TDD skill set* with respect to external quality and productivity.

Phase 1: Clustering of subjects

Rather than dividing our subjects according to arbitrary values of the components of the *TDD skill set*, we use the k-means [28] clustering algorithm to find the natural clusters in our dataset. The variables in the *TDD skill set* have different scales. Therefore, before clustering our dataset, we normalised the variables using z-scores.

Following the formula in [29] $\#clusters \approx \sqrt{\frac{n}{2}}$ (where n is the number of data points), we would expect between three and four clusters (n=30). Three appears to be an adequate choice, when the sum of square errors is plotted as a function of the number of clusters (Figure 2). We followed the elbow rule [30] and selected three clusters, since the differences of sum of square errors for a consecutive number of clusters do not fall significantly after three.

We used the k-means clustering algorithm [28], since it matches our assumptions:

1. We assume non-overlapping classes of subjects. (i.e., one subject cannot have two different levels of skills).

[1]We use this term to indicate that there was not true randomization when forming the groups

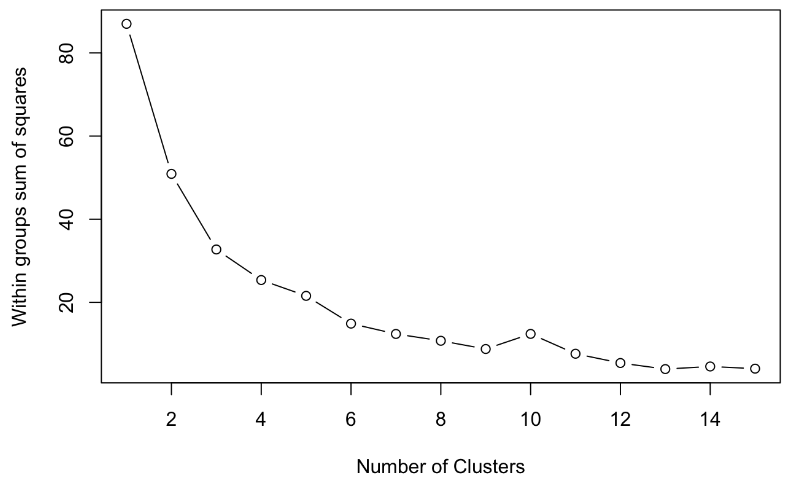


Figure 2: Plot of number of clusters as a function of within groups sum of squared errors, used to decide the a priori number of clusters

2. We are able to define a *prototype* subject for each class. (i.e., initializing the clusters' centroids).

Following assumption (2), the clustering algorithm yields the same solution after each run, since the initial centroids are fixed. The three hypothetical clusters can then be mapped onto three levels of *TDD skill set*: High, Medium, and Low.

In particular, the prototype of a subject having a *Low* level *TDD skill set* is represented, taking into account the unnormalized values, as $(UT = JAVA = CONF = 0)$, e.g., a subject with the lower possible value for each of the three skills. On the other hand, the prototype of a subject in the *High* level is represented as $(JAVA = UT = 3, CONF = 100)$, e.g., a subject with the highest possible value for each of the skills. Finally, the prototype for the *Medium* level is represented as $(JAVA = UT = 1.5, CONF = 50)$, e.g, a subject having skills equal to the central value for each of the three. Such prototypes, built by convenience, are used as the initial centroid for the k-means algorithm. The

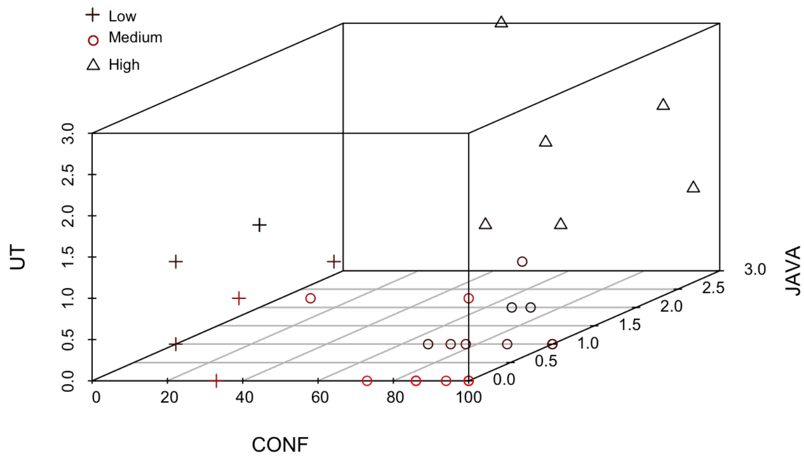


Figure 3: 3D scatterplot for UT, JAVA, and CONF with clusters. Please note that two data points overlap at ($CONF = 100, JAVA = UT = 0$); two data points overlap at ($CONF = 86, JAVA = UT = 0$); two data points overlap ($CONF = 0, JAVA = 2, UT = 1$).

result of the clustering is presented in Figure 3, where the obtained clusters can be visually identified by their labelling ($+$ = Low, $\bigcirc$ = medium, $\triangle$ = High).

Phase 2: Comparing subjects' clusters

Given our research questions, we want to check whether a linear dependency exists between a continuous response, i.e., QLTY and PROD, and a ordinal factor, *TDD skill set*, identified after clustering the subjects. In this context, rather than on prediction capability of *TDD skill set*, we focus on group differences. In order to execute the second phase, we use One-way Analysis of Variance (ANOVA) [31]. In particular, we are interested in comparing the means for QLTY and PROD in the three *TDD skill set* groups identified after Phase 1.

We focus on the two outcomes in isolation; therefore we prefer ANOVA over MANOVA (Multiple Analysis of Variance) [2] [31]. Since we do not possess any solid pre-existing knowledge about the relationship between the groups and the outcomes, nor an interest in the relationship between the two outcomes, the

[2]MANOVA allows to check the effect of groups on two outcomes *simultaneously*

MANOVA results would be hard to interpret and out of scope. Nevertheless, we acknowledge that, in further studies, a MANOVA design might offer a more holistic view of the effects of the *TDD skill set*.

The one-way ANOVA null hypothesis assumes that the groups are random samples from a population, and they all have the same effect on the outcome. ANOVA uses F-test to test whether the groups differ from each other or not. Hence, rejecting the ANOVA null hypothesis implies that the outcome is impacted by the group differences. When compared to multiple t-tests, ANOVA F-test is robust against false negative errors [31]. On the other hand, ANOVA only reveals whether there is a difference between *all* the groups, but not *where* the differences lie [31]. In our analysis—in order to check if interesting differences exists between *any* group—we also report the pairwise group comparisons as post-hoc analyses.

4.6. Hypotheses

Our research questions are mapped into statistical hypothesis testing by establishing whether a difference exists between the TDD skill groups in terms of the outcome. Observing a statistically significant difference between the groups implies an effect of skills.

We formally express research question RQ_{QLTY} with hypothesis H_{QLTY} as follows:

H_0 - $\mu(QLTY_{(High)})$=$\mu(QLTY_{(Medium)})$=$\mu(QLTY_{(Low)})$

H_1 - $\mu(QLTY_{(High)})\neq\mu(QLTY_{(Medium)})\neq\mu(QLTY_{(Low)})$

Correspondingly, we express research question RQ_{PROD} with hypothesis H_{PROD} as follows:

H_0 - $\mu(PROD_{(High)})$=$\mu(PROD_{(Medium)})$=$\mu(PROD_{(Low)})$

H_1 - $\mu(PROD_{(High)})\neq\mu(PROD_{(Medium)})\neq\mu(PROD_{(Low)})$

Please note that the three *TDD skill set* groups are labelled as *Low*, *Medium* and *High*, and μ represents the arithmetic mean.

Table 4: Summary of descriptive statistics for continuous variables $CONF$, $QLTY$, and $PROD$. (n=30).

variable	min.	1st qu.	median	3rd qu.	max.	mean	std.dev.
CONF	0.00	46.00	74.50	91.00	100.00	66.10	32.52
QLTY	26.28	39.61	46.43	55.35	100.00	48.96	15.74
PROD	3.03	14.40	22.35	26.33	40.91	21.06	9.36

5. Results

In this section, we first report the descriptive statistics of the data, and provide a sanity check in order to proceed with clustering and ANOVA. All the statistical tests use $\alpha = .05$.

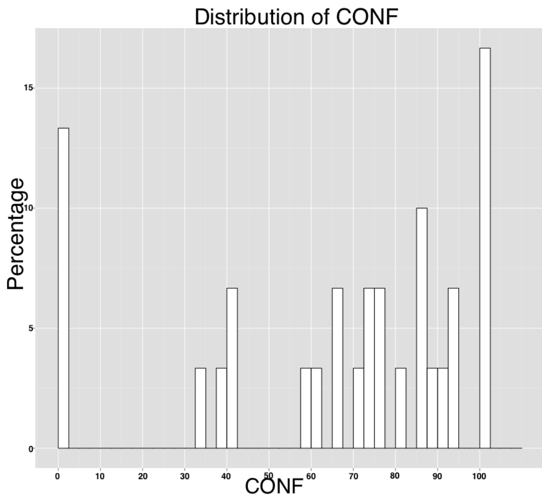
5.1. Descriptive Statistics

The descriptive statistics for the continuous variables, $CONF$, $QLTY$, and $PROD$ are reported in Table 4.

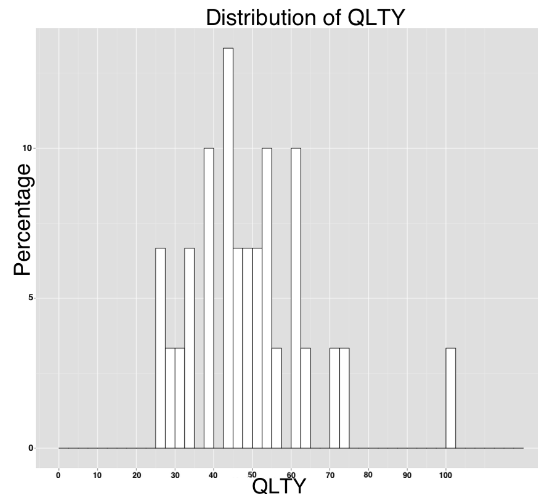
Process conformance ($CONF$). The mean, median, and 3rd quantile for $CONF$ suggest that the variable is left skewed, whereas the rather large standard deviation, compared to the mean, indicates that the data is dispersed. Figure 4a confirms the above, and shows a positive kurtosis. Although half of the subjects achieved at least 75% conformance, 13% seems to have relinquished TDD.

Software quality ($QLTY$). The distribution of $QLTY$ seems to follow the same properties of $CONF$ but with a right skew, as confirmed by inspecting Figure 4b. Around 58% of the subjects achieved a quality level between 26% (the sample minimum) and 48% (the sample mean). In the interval between 48% and 75%, we found almost all the remaining subjects, except for the remaining 4%, which are dispersed onto the maximum value (100%). Comparing the distributions for $CONF$ and $QLTY$, we notice that they are inverted.

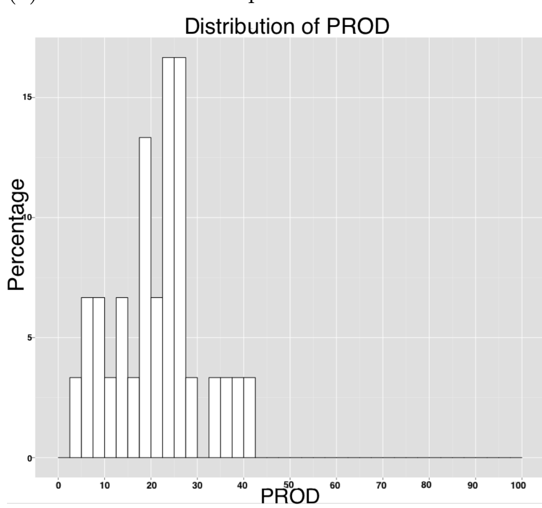
Developers' productivity ($PROD$). The distribution of $PROD$ (Figure 4c) is skewed right. The vast majority of the subjects (84%) could not implement



(a) Distribution of the process conformance variable



(b) Distribution of the external quality variable



(c) Distribution of the developers' productivity variable

Figure 4: Histograms representing the frequencies distribution for the continuous variables, $CONF$, $QLTY$, and $PROD$ (n=30)

more than one-third of the system[3], whereas the remaining 16% were able to

[3]This does not mean that they implemented the first task only, since the metric for PROD, reported in Formula 4, does not differentiate between tasks.

complete more than one-third but only up to the maximum level of 40%.

One preliminary observation that can be drawn from the distribution of $PROD$ is that the subjects implicitly decided to focus on a specific portion of the system. It seems that the settings of the study (i.e., near real-world complexity task together with time pressure) created a tradeoff between quality and productivity. When asked, in a post-questionnaire, for their opinion on the task, some subjects hinted that the choice of quality over productivity was deliberate. For example, one comment was:

> "I spent practically all of the exercise time improving the quality of the existing code. Although feature-complete, there was hardly any error handling and no unit tests. In my opinion, this code is not yet ready for production."

Java and unit testing skills ($JAVA$, UT). The subjects mostly identified themselves as having *None* or *Novice* Java programming skills, whereas only 10% claimed to have expert skills. This skewness towards the lower levels is more accentuated in the distribution of unit testing skills, for which only 3.4% of the subjects assessed themselves as expert, whereas more than half declared they did not have unit testing experience (Table 5). The two skills of interest

Table 5: Summary of distribution for ordinal variables $JAVA$, and UT (n=30).

	Levels			
Variable	None (0)	Novice (1)	Intermediate (2)	Expert (3)
JAVA	33.3%	33.3%	23.3%	10.0%
UT	53.4%	36.6%	6.6%	3.4%

are visualized through a mosaic plot in Figure 5. The width of each rectangle represent the proportion of subjects in the four levels of unit testing skill (UT), whereas the height represents the proportion of subjects in the four level of Java programming language skill ($JAVA$) (note that some rectangles might have only one dimension). Figure 5 represents how the two skills are distributed among each of the different levels (please note that the ordinal value have been replaced with their numeric counterpart). The subjects who self-identified as unit testing experts, also considered themselves Java experts. At the same time, all the Java experts agreed to have at least some experience with unit testing. The majority of the subjects with no Java skills also agreed that they did not possess unit testing skills.

Figure 5 reveals a partial similarity in the distribution of JAVA and UT, as there is an equal number of subjects for the JAVA *None* and *Novice* categories, as there are for the corresponding categories of UT. This might suggest that the two variables are correlated. The same information, in absolute terms, is reported in Table 6.

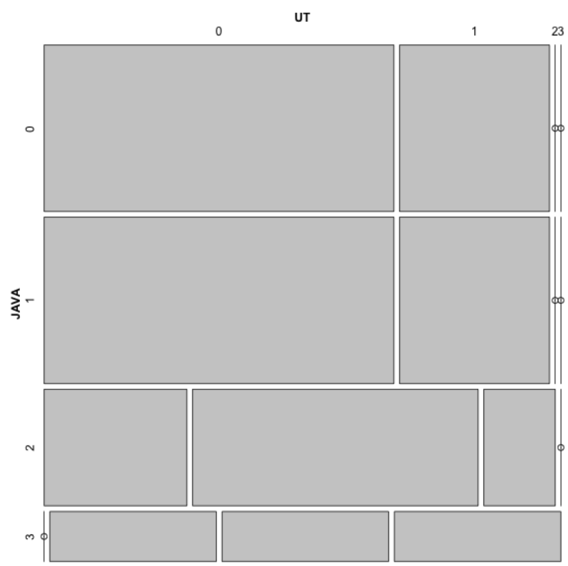


Figure 5: Distribution of the subject over the four levels of JAVA and UT. The four levels, *None*, *Novice*, *Intermediate* and *Expert* are mapped on the values from zero to three.

Table 6: Contingency table showing the distribution of subjects within the different levels of skills.

		JAVA			
		None	Novice	Intermediate	Expert
UT	None	7	3	0	0
	Novice	7	3	0	0
	Intermediate	2	4	1	0
	Expert	0	1	1	1

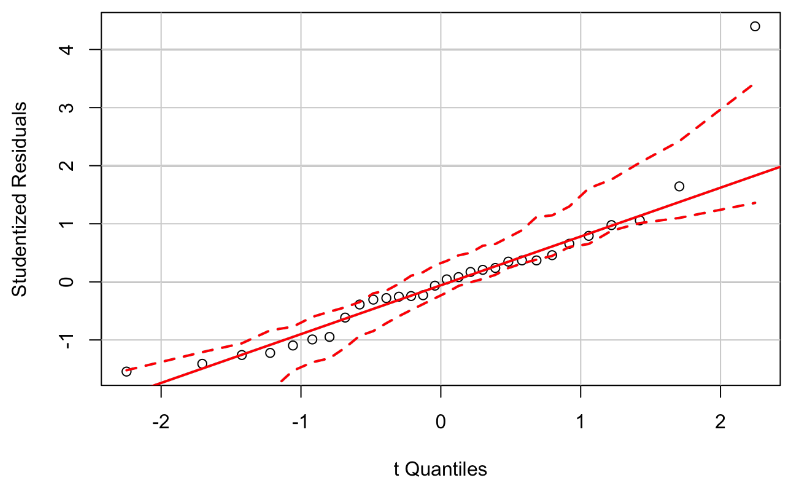
5.2. Diagnostics

Even though the clustering method we adopted does not require uncorrelated variables [28], we report the Spearman's correlation coefficients for our metrics in Table 7. For reference, please note that $JAVA$ and UT are positively correlated (Spearman $\rho = .5$, p=.004), as well as $PROD$ and UT (Spearman $\rho = .51$, p =.003)

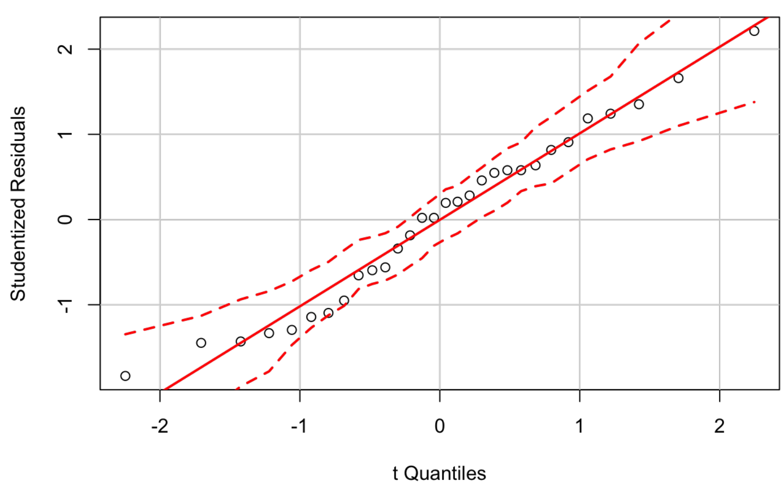
Table 7: Correlation table between variables.
(Spearman ρ in the matrix upper triangle, p-values in the lower triangle)

	CONF	QLTY	PROD	JAVA	UT
CONF		0.31	-0.14	-0.25	-0.30
QLTY	.093		0.46	0.20	0.29
PROD	.461	.009		0.36	0.51
JAVA	.179	.293	.049		0.50
UT	.108	.119	.003*	.004*	

In One-way ANOVA, the dependent variable is assumed to be normally distributed with equal variance in each group [32]. The Q-Q plot in Figure 6a shows that, for external quality, the normality is not met due to one data point (upper-right) that might be an outlier. Our concerns are confirmed by the Bonferroni test for outliers [33]. The outlier, also visible from Figure 4b, is characterized by the *TDD skill set* (CONF=100, JAVA=0, UT=0) and $QLTY$



(a) QLTY



(b) PROD

Figure 6: Q-Q plots used to check the normality assumption of ANOVA

Table 8: Summary of the ANOVA diagnostics. Bartlett K-squared test for equal variance and Bonferroni test for outliers results.
(*outlier removed)

Test (p-value)	QLTY		PROD
Bartlett K-squared	5.413 (.066)	*1.961 (.375)	0.218 (.896)
Bonferroni r-student	4.4 (.004)	*2.591 (.457)	2.213 (.358)

= 100. We believe that the outlier is not the result of a measurement error, but a legitimate case. This particular subject spent the entire time tackling a single feature of the task, and completing only that one with 100% quality. $QLTY$ depends on the $\#TUS$ (see Formula 2). $\#TUS$ descriptive statistics are: *min = 1*, *1st qu. = 5*, *median = 6*, *mean = 6.3*, *3r qu. = 8.75* and *max = 10*. It is apparent that in general the subjects tackled a substantial part of the system (on average more than half), except for the outlier. The shortcomings of the $QLTY$ metric are further discussed in Section 7. If we were to keep the outlier, we would need to correct the skewness of the distribution by apply some transformation procedure (e.g. square-root transformation). However, transformations make the interpretation of the results more complex and unreliable [34]. We prefer to remove the datapoint since it does not bring much information about the general level of $QLTY$. Please note that removal of the outlier did not introduce any new outliers.

The equality of variance is not violated ($p - value > 0.05$), as shown by Bartlett' K-squared test result [32] (Table 8).

The Q-Q plot for productivity (Figure 6b) shows that all the data points fall inside the 95% confidence interval. The variance is equal within each group, as shown by the Barlett K-squared test; and no significant outliers are present, as shown by Bonferroni test (Table 8).

After these sanity checks, the data meets the requirement for a robust ANOVA analysis.

5.3. Cluster Analysis

We analyse whether the three clusters of subjects differ in terms of quality and productivity. Table 9 reports the aggregate values of QLTY and PROD for the three clusters.

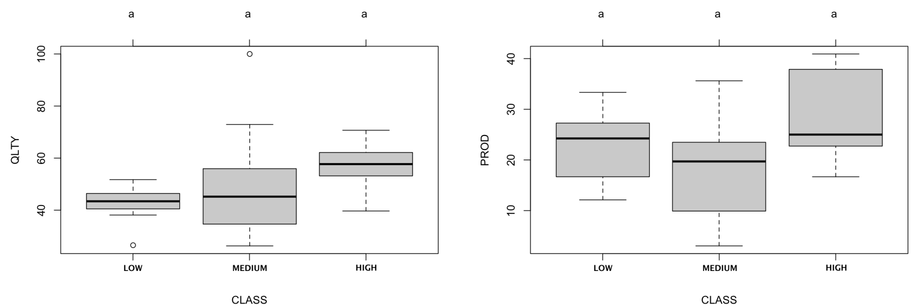
Table 9 and Figure 7a show that the level of quality achieved by the subjects in the *High* group is greater than the other two. The median lines in the boxplot (Figure 7a) shows that the subjects in the Low and Medium group performed equally. Although the height of the boxes and the standard deviations indicate that the performance of the Low group's subjects are more homogeneous. The height of the boxplots' sections in Figure 7a for the three groups are even around the median, suggesting that the distribution within each cluster follows a Gaussian curve. Moreover, notice the possible outliers below the lower whisker in the *Low* group and above the upper whisker in the *Medium* group.

Table 9: Summary of the distribution of QLTY and PROD for the three clusters

Symbol (Figure 3)	TDD skill set	n	QLTY		PROD	
			mean	sd	mean	sd
+	Low	6	42.22	8.09	22.51	7.74
∘	Medium	17	48.94	18.59	18.01	8.92
△	High	7	56.85	10.62	28.03	9.41

Regarding productivity, the subjects in the *High* group performed better than the other groups, but in this case, subjects in the *Medium* group were outperformed by those in the *Low* group, as shown in Figure 7b. The productivity of the three clusters of subjects tends to accumulate either below the median (*Low* and *Medium* groups) or above it (*High* group). The shape of the boxplots and the position of the whiskers does not indicate any substantial difference between groups.

One observation that is worth mentioning is that the baseline performances in the *High* group improves over the other groups for both external quality and productivity.



(a) QLTY (b) PROD

Figure 7: Boxplot for the three clusters

Table 10: Results of ANOVA for the levels of TDD skill set on QLTY and PROD. Note that the degrees of freedom for QLTY F-test are different from PROD after one outlier was removed.

	F-test	degrees of freedom	p-value
QLTY	1.44	(2, 26)	.260
PROD	3.02	(2, 27)	.065

Software quality (QLTY). We compared the QLTY of the three groups using ANOVA. A significant difference does not exist between the three groups of the *TDD skill set* in terms of external quality (Table 10). We fail to reject the null hypothesis in H_{QLTY}.

The *estimated* effect size ($\eta^2 = .09$, CI $= [0, 0.28]$), which can be interpreted as the percentage of variance of the external quality explained by the *TDD skill set*, in the case of our data, 9% is considered to be small-medium [35]. The non-significant p-value, confirmed by the confidence interval containing zero (reported in Equation 5, after the Bonferroni correction for multiple comparisons), does not allow a strong inference to be made about the effect on the actual population from which the sample was drawn. Nevertheless, we report effect size and confidence interval values in order to ease future meta-analyses attempts.

$$\begin{aligned}\mu(QLTY_{(high)}) - \mu(QLTY_{(low)}) &\in [-3.05, 32.31]\\ \mu(QLTY_{(high)}) - \mu(QLTY_{(medium)}) &\in [-7.18, 23.01]\\ \mu(QLTY_{(medium)}) - \mu(QLTY_{(low)}) &\in [-7.55, 20.99]\end{aligned} \tag{5}$$

Developers' productivity (PROD). The results of the ANOVA (Table 10) regarding productivity show that the null hypothesis in H_{PROD} failed to be rejected. In this case, we obtained a large *estimated* effect size ($\eta^2 = .18$, CI = [0, 0.38]), although the confidence interval supports the decision of not rejecting the null-hypothesis.

$$\begin{aligned}\mu(PROD_{(high)}) - \mu(PROD_{(low)}) &\in [-4.49, 15.53]\\ \mu(PROD_{(high)}) - \mu(PROD_{(medium)}) &\in [1.47, 18.56]\\ \mu(PROD_{(medium)}) - \mu(PROD_{(low)}) &\in [-12.58, 3.58]\end{aligned} \tag{6}$$

The null hypothesis for ANOVA is that the mean (average value of the dependent variable) is the same for all groups. The alternative or research hypothesis is that the average is not the same for *all* groups. The pair-wise analysis of the confidence intervals, after the Bonferroni correction for multiple comparisons, reported in Equation 6 shows that a significant difference exists between *High* and *Medium* levels of the *TDD skill set* (i.e., the confidence interval does not contain zero).

6. Discussion

We investigated two research hypotheses in which we argue that a difference in terms of external quality (H_{QLTY}) and productivity (H_{PROD}) exists among three *TDD skill set* groups.Our TDD skill set includes two different kind of skills: *a-priori* knowledge of concepts necessary to apply TDD (i.e., Java programming language and unit-testing); and *in process* skill, i.e., the level of conformance to the TDD process. We first clustered the subjects according to their skills' set, then we applied statistical hypotheses testing based on ANOVA.

We do not have evidence to assert that either null hypotheses are false; however this does not imply that they are true. Based only the result of statistical hypotheses testing, i.e., *p-value*, our results appear to be inconclusive. Nevertheless, we discuss the implications of the effect sizes and confidence intervals (CIs) to provide more insight. Although—in statistical terms—we can not accept the null hypothesis, CIs tell us if the differences between the skill set groups would likely be meaningful or not. In other words, we base our conclusion on whether the true deviation from the null hypothesis is too small to worry about. We prefer to follow this line of argument rather that the simple *reject/fail to reject* dichotomy of hypothesis testing [36, 37].

Moreover, we report the 95% CI around the effect sizes which provide a standardized metric allowing for comparisons across studies. Also the conclusions based on such CIs do not focus on whether the null hypothesis is viable; rather they represent how large a deviation from the null hypotheses we can reasonably expect in the population.

With this line of reasoning, the answer to the research question(s) are:

> Although our data did not show a statistically significant difference between the developers' *TDD skill set* and external quality or productivity, the skill set's effect is worth further investigation as it could bring substantial improvement for both outcomes.

The rationale and the implications of the answer to the research question is given below.

External quality. In our previous studies [9, 16], we observed a positive, although not significant, trend between the conformance component of the TDD skill [16] and external quality. We observed the same results when studying the Java and unit-testing skills [9].

The pair-wise comparison between the groups shows that skilled developers (labelled as *High*) are able to reach better quality levels with respect to the others (labelled as *Medium* and *Low*).

In fact, the CI around the differences between the means of the *High* and *Low* can be as much as 32.21 points in favor of the first group. Considering that $QLTY \in [0, 100]$, this is a substantial 30% difference. In the opposite case, the difference in favor of the *Low* group can be only as much as 3.05. The CI provides support against the null hypotheses (i.e., the difference is zero) as a sizeable difference exists in the direction of the *High* group. The same applies for the other groups' comparison. The better ranked group can perform as much as around 20% better than the other.

The CI around the effect size—$\eta^2 = .09$, $CI = [0, 0.28]$—reported for the ANOVA—tells that the population effect could be as large as 0.28, which is considered *large* [35]. In other words, the TDD skill set can account for up to 28% of the change in external quality. Note that the effect size CI contains zero, i.e., there might be no effect of TDD skill set on the population. This is consistent with the hypothesis test result and the CIs for the pair-wise comparison. The effect size should be interpreted with caution since its estimation for the population can be biased.

We failed to reject the null hypotheses in H_{QLTY}. This present two possible alternatives:

a) The null hypothesis is actually true in reality, i.e., there is no real effect of the TDD skill set on the external quality

b) The null hypothesis is false, but we cannot reject it. This reflects a lack of power in our study and a Type II error.

Following alternative *a*) would imply in practise that, for example, external quality cannot be significantly improved by training developers on the skills in our skill set. As the skill set focuses on technical abilities, one development from this result would be conducting a similar study investigating the role of *soft* skills instead, like the ability of the developers to understand the requirement, developers' creativity [38], and other psychological factors [39].

Nevertheless, given the analysis of the effect size and CI presented here, we believe that this study falls under the alternative *b*): we are witnessing a false negative. This means that, although our data did not show any effect of skills, such effect is actually present in reality. In this case, further actions should focus on avoiding possible Type II error and improve the chances to find an effect where there is one. For further replication we suggest a sample size of at least 63 subjects (21 subjects per group) [35]. The sample size is the result of a power analysis for ANOVA design study (F-test family), with the typical values for $\alpha = 5\%$ and $\beta = 20\%$ [40] to yield a *large* effect size, as indicated by the CI of the estimated effect size reported in this study.

Note that we do not discuss Type I error, or false positive—i.e., witnessing the effect of skills in our data where such effect is not present in reality—since that would require the rejection of the null hypotheses in the first place.

Finally, although this study did not achieve statistical significance, there is evidence that the study of the effects of TDD skills on external quality is worth pursuing. A similar study [11]—in which the subjects were not divided using clustering, but according to their level of seniority—provides some evidence that skill or experience could improve external quality when TDD is employed. Whilst a single primary study like this one can rarely provide clear-cut advices, we put forward the idea that developers should aim for an high level of TDD skills in order to produce software with better external quality.

Developers' productivity. The ANOVA result shows that only for 6.5% of the time, the differences between the groups are down to chance, whereas 18% of the productivity variance can be explained by the *TDD skill set*. From the pair-wise comparison (Equation 6), we show that a significant difference only exists between the *High* and *Medium* groups. In particular this difference between the means can be substantial, up to 18.56 points on a scale of one hundred, as showed in the CI.

What we also observed is that the order of the groups does not follow the expected one (*High* >*Low* >*Medium* rather than *High* >*Medium* >*Low*). Moreover, the subjects are spread within each group but not in absolute terms. The *Medium* and *Low* groups might have much in common, based on their overlap 7b and the range of the CI for means difference. Also, the CI for the difference between the means of *High* and *Low* groups is not accentuated in either direction, in contrast to $QLTY$.

Nevertheless, the ANOVA estimated effect size and CI (η^2 = .18, CI = [0, 0.38]) indicate that the study of the skill set should be further pursued, since almost 40% of the developers' productivity might be due to it. Also for the case of productivity we believe that a true effect is present in reality but our study sample is just not large enough. Therefore the same suggestion provided for further studies of external quality applies to productivity.

7. Threats to Validity

In this section, we explain the main threats to the validity of our study following Wohlin et al. [41], along with the countermeasures we took when possible. Moreover, we suggest some actions that researchers willing to replicate this study could take to limit some of the threats. The types of validity threats are prioritized, in increasing order, following Cook and Campbell's [42] guidelines. In particular, since this study is part of an effort to apply research in industry, we give more importance to generalizability.

7.1. External Validity

We believe that the task (non-toy and brownfield) used and the subjects (professionals from industry) participating in this study are a good approximation of the reality. Nevertheless, we acknowledge some limitations related to two elements: domain and duration. The study was carried out in the same environment and working hours the subjects are used to but, in their daily work, they deal with two very specific domains: security and gaming. While we did not consider such differences, we recommend that future attempts to replicate this study consider these differences to provide more generalizable results. We conclude that our results might not be generalisable to software developers in other domains. The duration of the study also limits its generalizability. For applied research, it is important to target scenarios that are a good approximation of the real world, but the researcher community struggles with the cost and scheduling issues of running long-term studies within a company [43]. Yet, initiatives like this one are useful for the advancement of the research field, since they might reveal the presence of hidden constructs that could better explain the phenomenon under study, or the inadequacy of the generally accepted constructs in use [44].

7.2. Construct Validity

The principal threats are related to the design of the study. A mono-operation bias might occur, since only one task was used to measure the outcomes. Although, due to logistic reasons, it was not possible to prevent this threat, we suggest that future replications of this study use more than one task and include it as a covariate in their model.

At the same time, the study suffers from a mono-method bias threat, since only one metric was used to measure the constructs. We acknowledge that the variables with which we operated can be measured in several ways; however, we selected metrics that have been used in several previous studies [23, 45, 46].

The interaction of testing and treatment might have occurred, since the workshop in which the subjects took part was about TDD and unit testing, so they were aware of the importance of applying TDD during the study. This might have inflated the measurements of process conformance to the detriment of, for example, productivity. In other words, the subjects were primed in the thorough application of TDD and might have decided to make it their focus rather than complete the task. However, our data shows a diverse set of process conformance values, even indicating that some subjects did not use TDD at all.

We are aware that a restricted generalizability across constructs might exist because we neglected other constructs that might be affected during the study, such as internal quality or maintainability. Although this specific study did not show any substantial effect, we advise that other outcomes of potential interest should be observed for future studies.

The concepts underlying the constructs used for this study appear to be clear enough to not constitute a threat. In particular, the TDD cycle is defined using the existing literature, and the metrics associated with it uses empirically validated operational definitions [23].

There are no significant social threats to construct validity. Specifically, evaluation apprehension–the tendency of fearing being evaluated–should not have an impact, since the subjects participated in the study on a voluntary basis, and it was agreed that our results would be shared with their employers only in aggregated forms.

In addition, we do not believe that hypotheses guessing poses a threat since the research questions were not disclosed, albeit the participants were aware they were taking part in a research trial. Finally, we are aware that the Hawthorne effect [47] might have taken place, since the subjects were observed by several researchers throughout the study. Nevertheless, the subjects were accustomed to this condition because the researchers were present throughout the workshop.

7.3. Conclusion Validity

We are concerned with the ability to draw the correct conclusion from the results of our tests. In fact, when the null hypothesis is not rejected, there is the possibility to commit a Type II error. The use of ANOVA without covariates other than the *TDD skill set* might have decreased the statistical power of the test, since there are no other factors accountable for the unexplained variance of the dependent variables. However, the validity of our present conclusions are strengthened by validating the test assumptions detailed in Section 5.2.

Another concern is the reliability of measures. In particular, Java programming and unit testing skills were measured using subjective measures, which are prone to the specific subject's biases [48]. Nevertheless, a review of how knowledge and skills of developers are measured in controlled experiments, found that self-assement to be a reliable way to measure such constructs [26]. The objectivity of the process conformance metric is guaranteed by the fact that it is calculated without, or with little, human intervention through a software tool [48]. A possible bias can be introduced in the *QLTY* metric once a subject starts working on a user story right before the end of the allotted time, rather then decide to stop working. Consider Table 11. Subject A completes the first three user-stories with a $QLTY$ of 60% and decides to stop working since the time is approaching the end of the session. Subject B is in the same situation of Subject A, but decided to tackled the next user story, she runs out of time and she is able to deliver US4 only at 20% of its $QLTY$. The final $QLTY$ score for Subject B is then 50%.

Table 11: Example of possible bias in QLTY.

Subject	QLTY(US1)	QLTY(US2)	QLTY(US3)	QLTY(US4)	QLTY
A	0.7	0.5	0.6	not tackled	0.6
B	0.7	0.5	0.6	0.2	0.5

However, the subjects were not aware of the user-stories and they could only see the sub-tasks presented as in Appendix C. Yet, by chance, a subjects could have inadvertently started working on a user-story and run out of time. We acknowledge that this could have be disadvantageous for some subjects, although all of them were aware of the time limit given to complete the task.

The random heterogeneity of sample threat might have occurred, since our subjects ranged from having a few months to more than 10 years experience in software development. Nevertheless, we do not consider this a major problem for two distinct reasons: first, participation in the workshop was voluntary, as we could not ask the companies for a specific demographic; second, we prefer to not emphasize this threat and leverage the trade-off of having a better generalizability.

We limited the implementation threat by dedicating part of the workshop to explain and apply TDD in detail, and we reminded the subjects during the study to follow the guidelines we gave them. We do not believe that fishing jeopardised the study, since none of the researchers or the companies' representatives that helped set up the study had any specific expectations about the results. Finally, no external extraordinary events—that might interfere with the execution of the study (e.g., the CEO of the company visiting on site, an earthquake)—took place.

7.4. Internal Validity

We run a quasi-experiment, since it is not possible to randomly divide the subjects according to their *TDD skill set*. Nevertheless, a design in which the subjects are divided according to their process conformance, based on the result of a pre-test, might be a solution. In turns, such design has the shortcoming of injecting a confounding effect due to the use of different tasks (one for the pre-test, one for the experiment), and a carry-over effect once the tasks are too similar.

In general, a quasi-experimental design does not allow to make strong causation inference [17], as the direction of causality can be difficult to assess. We pointed out that the *TDD skill set* might have an impact on external quality and productivity, but the direction of the relationship can be, hypothetically, reversed. However, this is unlikely in the settings of this study due to the nature of the observation—i.e., external quality and productivity are observable only after the development process is finished.

An important validity threat to the study is the maturation process that might have taken place among the subjects. In particular, the subjects' Java programming and unit testing skills were measured before the study took place. After that, the subjects participated in a workshop in which Java and unit testing concepts were used. This means that some subjects' initial skills might have changed before the study took place. We advise that future replications of this study should promote a pre-post assessment of the subjects, possibly using objective measures that can be easily compared.

In addition, the sample might not be representative of the population. The participation in the workshop was voluntary, although the subjects did not have previous training on TDD. We believe that the low unit testing experience is also due to the subject recruitment process. The workshop was attractive for engineers willing to learn unit testing and TDD, leaving out the more expert ones.

A minor threat to the internal validity might arise from instrumentation issues. In particular, the subjects performed the study task in an environment (e.g., operating system, integrated development environment) with which they were not necessarily familiar. On the other hand, such a threat is mitigated by having the subject use the same environment during the workshop.

Finally, although the overall mortality was 30% (i.e., 13 subjects of the 43 sampled were removed), only 5% was caused by subjects intentionally leaving the study, and the remaining 25% was due to technical issues. The specific problem was related to the execution of the tool that we used for measuring process conformance [22] in a virtual machine. Although we communicated the baseline hardware requirements beforehand, it was not possible to upgrade participant's computers in the field. We suggest, for future replications to provide the subjects with a sandbox environment, for example a virtual machine like we did, but to make sure to emphasise that the subjects' machine matches the hardware requirements in which the tool is successfully tested.

8. Conclusion

In this work, we studied 30 professional software developers applying TDD to add new features to a legacy system close to real-world complexity. We contributed to the existing knowledge by operationalising developers' test-driven development skills, not only according to their *a priori* abilities (i.e., Java programming and unit testing), but also including their capacity to follow the test-driven development cycle. We clustered the subjects according to such skill set and compared them in terms of external quality and productivity. We found that no significant difference exists between the groups.

A deeper analysis of the the pair-wise difference between the groups' means, the ANOVA estimated effect sizes, and their 95% CIs shows that we might be committing a Type II error. In fact, for both outcomes, a real effect of the *TDD skill set* might exists in reality, but the power of our study is not enough to reveal it. We suggest to replicate this study with a sample of, at least, 63.

The reported effect sizes and their CIs give us reason to believe that the *TDD skill set* can account for up to 28% the variance in external quality, and 38% in productivity. Hence, *TDD skill set* might be a valuable asset for developers.

Further studies are required to assess how developers' skills impact external quality and productivity. As Thomson and McConnel needed around 400 replications of their experiment to uncover the factors involved in flatworms' memory transfer [49], this study is only a part of larger process to uncover moderators and hidden variables necessary to understand TDD and control it under experimental settings.

We foster replications in order to assure correct conclusions [49] and to address the shortcomings we have identified in Section 7. For example:

- Control for subjects maturation during the training
- Include subjects from other domains
- Collect data over a longer time period
- Collect data from different types of tasks
- Use different metrics to measure the constructs
- Measure skills objectively (e.g., using pre-tests)

We also recommend future study to investigate how *soft* skills related to TDD impact the same outcomes presented in this paper. A lab package for this study is available for future replications [50].

Acknowledgements

This research is partially supported by the Academy of Finland with decision no: 278354, and by Finnish Distinguished Professor (Fi.Di.Pro.) programme, ESEIL. The first author would like to acknowledge the Nokia Foundation and ISACA Finland chapter for the support provided in completing this work. We would like to acknowledge Dr. Lucas Layman who significantly contributed to the design of the task used in the study. We would like to acknowledge the anonymous reviewers for their helpful comments and detailed suggestions.

References

[1] K. Beck, Test-driven Development: by Example, 1st Edition, The Addison-Wesley signature series, Addison-Wesley Professional, 2002.

[2] D. Astels, Test Driven development: A Practical Guide, 2nd Edition, Prentice Hall Professional Technical Reference, 2003.

[3] K. Beck, Aim, fire, IEEE Software 18 (5) (2001) 87–89.

[4] Y. Rafique, V. Misic, The effects of test-driven development on external quality and productivity: a meta-analysis, IEEE Transactions on Software Engineering 39 (6) (2013) 835–856.

[5] B. Turhan, L. Layman, M. Diep, H. Erdogmus, F. Shull, How Effective Is Test Driven Development?, in: A. Oram, T. Martin (Eds.), Making Software: What Really Works, and Why We Believe It, 1st Edition, O'Reilly Media, 2010, Ch. 12, pp. 207–217.

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

[7] D. Fucci, B. Turhan, M. Oivo, On the effects of programming and testing skills on external quality and productivity in a test-driven development context, in: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, no. 25 in EASE '15, ACM, New York, NY, USA, 2015, pp. 1–6.

[8] D. Fucci, B. Turhan, On the role of tests in test-driven development: a differentiated and partial replication, Empirical Software Engineering (2013) 1–26.

[9] D. Fucci, B. Turhan, M. Oivo, Impact of process conformance on the effects of test-driven development, in: the 8th ACM/IEEE International Symposium, no. 10 in ESEM '14, ACM Press, New York, New York, USA, 2014, pp. 1–10.

[10] H. Munir, M. Moayyed, K. Petersen, Considering rigor and relevance when evaluating test driven development: A systematic review, Information and Software Technology 56 (4) (2014) 375–394.

[11] R. Latorre, Effects of developer experience on learning and applying unit test-driven development, Software Engineering, IEEE Transactions on 40 (4) (2014) 381–395. doi:10.1109/TSE.2013.2295827.

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

[13] A. T. Misirli, H. Erdogmus, N. Juristo, O. Dieste, Topic selection in industry experiments, in: Proceedings of the 2nd International Workshop on Conducting Empirical Studies in Industry, CESI 2014, ACM, New York, NY, USA, 2014, pp. 25–30. doi:10.1145/2593690.2593691.

[14] I. Salman, A. Tosun Misirli, N. Juristo, Are students representatives of professionals in software engineering experiments?, in: Proceedings of the 37th International Conference on Software Engineering, ICSE 2015, ACM, New York, NY, USA, 2014.

[15] L. A. Meyerovich, A. S. Rabkin, Empirical analysis of programming language adoption, SIGPLAN Not. 48 (10) (2013) 1–18. doi:10.1145/2544173.2509515.

[16] D. Fucci, B. Turhan, M. Oivo, Conformance factor in test-driven development: Initial results from an enhanced replication, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, no. 22 in EASE '14, ACM, New York, NY, USA, 2014, pp. 1–4. doi:10.1145/2601248.2601272.

[17] W. R. Shadish, T. D. Cook, D. T. Campbell, Experimental and Quasi-Experimental Designs for Generalized Causal Inference, 2nd Edition, Cengage Learning, 2003.

[18] M. Weinberger, E. Z. Oddone, W. G. Henderson, D. M. Smith, J. Huey, A. Giobbie-Hurder, J. R. Feussner, Multisite randomized controlled trials in health services research: scientific challenges and operational issues, Medical care 39 (6) (2001) 627–634.

[19] S. W. Raudenbush, X. Liu, Statistical power and optimal design for multisite randomized trials., Psychological methods 5 (2) (2000) 199.

[20] B. Vodde, L. Koskela, Learning test-driven development by counting lines, Software, IEEE 24 (3) (2007) 74–79. `doi:10.1109/MS.2007.80`.

[21] D. Sato, H. Corbucci, M. Bravo, Coding dojo: An environment for learning and sharing agile practices, in: Agile, 2008. AGILE '08. Conference, 2008, pp. 459–464. `doi:10.1109/Agile.2008.11`.

[22] K. Becker, B. de Souza Costa Pedroso, M. S. Pimenta, R. P. Jacobi, Besouro: A framework for exploring compliance rules in automatic {TDD} behavior assessment, Information and Software Technology 57 (0) (2015) 494 – 508. `doi:http://dx.doi.org/10.1016/j.infsof.2014.06.003`.

[23] H. Kou, P. Johnson, H. Erdogmus, Operational Definition and Automated Inference of Test-driven Development with Zorro, Automated Software Engineering 17 (1) (2010) 57–85. `doi:10.1007/s10515-009-0058-8`.

[24] Y. Wang, H. Erdogmus, The role of process measurement in test-driven development, in: C. Zannier, H. Erdogmus, L. Lindstrom (Eds.), Extreme Programming and Agile Methods - XP/Agile Universe 2004, Vol. 3134 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, pp. 32–42. `doi:10.1007/978-3-540-27777-4_4`.

[25] N. Zazworka, V. R. Basili, F. Shull, Tool supported detection and judgment of nonconformance in process execution, in: Empirical Software Engineer-

ing and Measurement, 2009. ESEM 2009. 3rd International Symposium on, IEEE, 2009, pp. 312–323.

[26] J. Siegmund, C. Kästner, J. Liebig, S. Apel, S. Hanenberg, Measuring and modeling programming experience, Empirical Software Engineering 19 (5) (2014) 1299–1334. doi:10.1007/s10664-013-9286-4.

[27] G. R. Bergersen, D. I. K. Sjoberg, T. Dybå, Construction and Validation of an Instrument for Measuring Programming Skill, Software Engineering, IEEE Transactions on 40 (12) (2014) 1163–1184.

[28] P.-N. Tan, M. Steinbach, V. Kumar, Cluster analysis: basic concepts and algorithms, in: Introduction to Data Mining, 1st Edition, Pearson, 2005, pp. 487–568.

[29] K. V. Mardia, J. Kent, J. Bibby, Multivariate Analysis, 1st Edition, Probability and Mathematical Statistics, Academic Press, 1980.

[30] M. Yan, K. Ye, Determining the number of clusters using the weighted gap statistic, Biometrics 63 (4) (2007) 1031–1037. doi:10.1111/j.1541-0420.2007.00784.x.

[31] D. C. Montgomery, D. C. Montgomery, D. C. Montgomery, Design and analysis of experiments, Vol. 7, Wiley New York, 2001.

[32] G. Snedecor, Statistical methods, 8th Edition, Iowa State University Press, 1989.

[33] R. D. Cook, P. Prescott, On the accuracy of Bonferroni significance levels for detecting outliers in linear models, Technometrics 23 (1) (1981) 59–63.

[34] J. W. Osborne, Improving your data transformations: Applying the Box-Cox transformation, Practical Assessment, Research & Evaluation 15 (12) (2010) 1–9.

[35] C. O. Fritz, P. E. Morris, R. J. J, Effect size estimates: Current use, calculations, and interpretation, Journal of Experimental Psychology: General 141 (1) (2012) 2–18.

[36] G. R. Loftus, Psychology will be a much better science when we change the way we analyze data, Current directions in psychological science (1996) 161–171.

[37] B. Thompson, What future quantitative social science research could look like: Confidence intervals for effect sizes, Educational Researcher 31 (3) (2002) 25–32.

[38] R. Mohanani, P. Ralph, B. Shreeve, Requirements fixation, in: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, ACM, New York, NY, USA, 2014, pp. 895–906. doi:10.1145/2568225.2568235.

[39] D. Graziotin, X. Wang, P. Abrahamsson, Are happy developers more productive?, in: J. Heidrich, M. Oivo, A. Jedlitschka, M. Baldassarre (Eds.), Product-Focused Software Process Improvement, Vol. 7983 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 50–64. doi:10.1007/978-3-642-39259-7_7.

[40] P. D. Ellis, The essential guide to effect sizes: Statistical power, meta-analysis, and the interpretation of research results, Cambridge University Press, 2010.

[41] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, 1st Edition, Computer Science, Springer-Verlag Berlin Heidelberg, 2012.
URL http://books.google.fi/books?id=QPVsM1_U8nkC

[42] T. D. Cook, D. T. Campbell, A. Day, Quasi-experimentation: Design and analysis issues for field settings, 1st Edition, Houghton Mifflin Boston, 1979.

[43] O. Dieste, N. Juristo, M. D. Martinez, Software industry experiments: A systematic literature review, in: Conducting Empirical Studies in Industry (CESI), 2013 1st International Workshop on, IEEE, 2013, pp. 2–8.

[44] B. Kitchenham, T. Dyba, M. Jorgensen, Evidence-based software engineering, in: Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on, 2004, pp. 273–281. doi:10.1109/ICSE.2004.1317449.

[45] H. Erdogmus, M. Morisio, M. Torchiano, On the Effectiveness of the Test-First Approach to Programming, IEEE Transactions on Software Engineering 31 (3) (2005) 226–237.

[46] L. Madeyski, Test-driven development: An empirical evaluation of agile practice, 1st Edition, Springer, 2009.

[47] J. G. Adair, The Hawthorne effect: A reconsideration of the methodological artifact., Journal of Applied Psychology 69 (2) (1984) 334.

[48] T. Lethbridge, S. Sim, J. Singer, Studying software engineers: Data collection techniques for software field studies, Empirical Software Engineering 10 (3) (2005) 311–341. doi:10.1007/s10664-005-1290-x.

[49] P. Diesing, How does social science work?: reflections on practice, University of Pittsburgh Press, 1991.

[50] D. Fucci, A lab package for TDD experiment replication (02 2015). URL http://dx.doi.org/10.6084/m9.figshare.1304256

Sudoku verifier

Sudoku is a game with few simple rules, where the goal is to place nine sets of positive digits (1...9) into the cells of a fixed grid structure (i.e. board).

The Sudoku board (or global grid) consists of a 3x3 arrangement of sub-grids, and each sub-grid is a 3x3 arrangement of cells. This yields a 9x9 arrangement of cells on the Sudoku board.

A valid Sudoku solution should conform to the following rules:

- R1: A cell in a Sudoku game can only store positive digits, i.e. 1...9.
- R2: All digits appear only once in a sub-grid, i.e. they cannot repeat.
- R3: A digit can appear only once in the rows of the global grid.
- R4: A digit can appear only once in the columns of the global grid.

Your task is to check the validity of a given solution for a Sudoku game

- You should read the candidate solution from a string variable, which should be exactly 81 characters long, i.e. first 9 are the first row, second 9 are the second row etc.

- You shall check whether the provided string follows the correct format (i.e. 9 rows with 9 entries in each row).

- You shall check the validity of the candidate solution against the rules listed above.

Figure A.8: One of the kata exercise used during the workshop.

Appendix B.

MusicPhone Tasks

MusicPhone is an application that runs on a GPS-enabled, MP3-capable mobile phone. MusicPhone will make recommendations for artists the user may like and find upcoming concert events for artists using data gathered from the Last.FM website. The goal of the following tasks is to implement the necessary logic to enable MusicPhone to make recommendations and plan a travel itinerary to concert events.

Task A: Ramp-up

A1. Run the project (Select the project, right click and select Run As-> Run Configurations. From the configurations, select *MusicPhone* from C/C++ Application) and see three UI windows appear. The Player and GPS UIs are complete. The Recommender window is just a skeleton. You will implement most of the required functionality in the `Recommender` class.

A2. Run the project with Unit Test configuration (Select the project, right click and Run As->Run Configurations. From the configurations, select *MusicPhone UnitTest* from C/C++ Unit) and see the green bar. Check the structure of the sample test in SmokeTest.cpp to get familiar with the application.

A2. Take 5-10 minutes to review the information in the provided documentation.

Task B: Compute distance to a concert

The user will see how far away upcoming concert events are for a particular artist based on the user's current GPS position and the position of the concert's venue. Implement the `Distance` class in `commons.dataClasses` and a public method with the signature `public static double computeDistance (GeoPoint, GeoPoint, String)` in this class to calculate the great-circle distance between two geo-coded positions. A geo-coded position is represented by a `GeoPoint` object that specifies a latitude and longitude *in degrees*. The method computes the great-circle distance between two points in either kilometres ("km") or miles ("mi") as specified by the third parameter. This parameter can be lower case, upper case or a combination of those. Valid latitude values are between -90 and 90 *degrees*; valid longitude values are between -180 and 180 *degrees*. Sometimes a `GeoPoint` has an invalid latitude or longitude. In these cases, the distance returned should be `-1`.

Examples:

GeoPoint₁		*GeoPoint₂*		*units*	**Great-circle distance** (in *units*)
$LatR_1$ $LatD_1$	*$LonR_1$* $LonD_1$	*$LatR_2$* $LatD_2$	*$LonR_2$* $LonD_2$		
0 0	*0* 0	*0* 0	*0* 0	km	**0**
0 0	*0* 0	*1.047* 60	*0* 0	km	**6671.70**
0 0	*0* 0	*6.283* 360	*6.283* 360	mi	**-1**
0 0	*0* 0	*1.047* 60	*0* 0	mi	**4145.60**
0.630 36.12	*-1.513* -86.67	*0.592* 33.94	*-2.066* -118.40	mi	**1793.55**

Numbers in roman type are in degrees.
Numbers in italics are in radians.

The formula for the great-circle distance is:

$LonD$ = longitude in degrees;
$LatD$ = latitude in degrees;
$LonR$ = longitude in radians;
$LatR$ = latitude in radians;

$$radians = (degrees \times \pi)/180;$$

$$a = \sqrt{\sin^2\left(\frac{\Delta LatR}{2}\right) + \cos(LatR_1) \times \cos(LatR_2) \times \sin^2\left(\frac{\Delta LonR}{2}\right)};$$

$$\text{Great Circle Distance} = 2 \times \arcsin(\min(1.0, a)) \times Radius$$

; where *Radius* is the Earth's radius in kilometers (6371.01) or in miles (3958.76).

Task C: Find concerts for an artist

Implement the `getDestinationsForArtist` method of the `Recommender` class. When the user clicks on an artist's name in the list of recommended artists, the UI calls the `getDestinationsForArtist` method to obtain a list of destinations, where each destination contains the info of an upcoming concert together with the distance to the concert's location from the user's current position. The user's current position is provided by the `GPS` class. If the concert's location has an invalid `GeoPoint`, then the distance should be marked as `-1`. You'll need to use `IConnector's GetConcertForArtist` method to implement this task.

Task D: Recommend artists

Implement the `getRecommendations` method of the `Recommender` class. MusicPhone recommends artists based on the favourite artists of Last.FM users who are the top fans of the currently-playing artist on the user's player. MusicPhone recommends the 20 most-frequently-appearing artists among those fans' top artists (as in "people who like this artist also like these other artists..."). `getRecommendations` should return a list of Recommendation objects. Each `Recommendation` object consists of an artist's name (`artist`) and the frequency of occurrence (`fanCount`) of that artist. You will need to use `IConnector's getTopFansForArtist and getTopArtistsByFan` methods to implement this method.

Example: Suppose the currently playing artist is *Cher*. Suppose *John*, *Sally*, and *Tom* are the only top fans of *Cher*. If *Tom* and *John* both like *U2* as well, then `getRecommendations` should return a list that includes a recommendation containing *U2* with a `FanCount` of 2.

Task E: Compute an itinerary for concerts

E0. Implement the `buildItineraryForArtists` method of the `Recommender` class. In the UI, the user can click on buttons that will add (Add button) or remove (Remove button) a selected artist to a list of artists who the user would like to see in concert. Whenever this list is changed, the application recalculates an itinerary of upcoming concerts by calling this method. An itinerary is a list of `Destination` objects, where each object contains the concert information and the distance to the concert's location from the previous destination. The first destination's distance in the list is the distance from the user's current position. The itinerary must be chronologically ordered according the the start date of the concerts in it. Satisfy the following constraints in any order, but tackling them *one by one:*

E1. The itinerary starts with the concert having the earliest start date for any of the selected artists.
E2. An artist has at most one concert in the itinerary (some artists may have no concerts, however).
E3. No two concerts in the itinerary may take place on the same day.
E4. If multiple artists have concerts on the same day,the itinerary includes only the concert that is closest in distance to the previous destination in the list.
E5. If the same artist has multiple concerts on the same day, the itinerary inlcudes only the concert that is closest in distance to the previous destination in the list.

Figure B.9: Instructions for the task used in this study.

Appendix C.

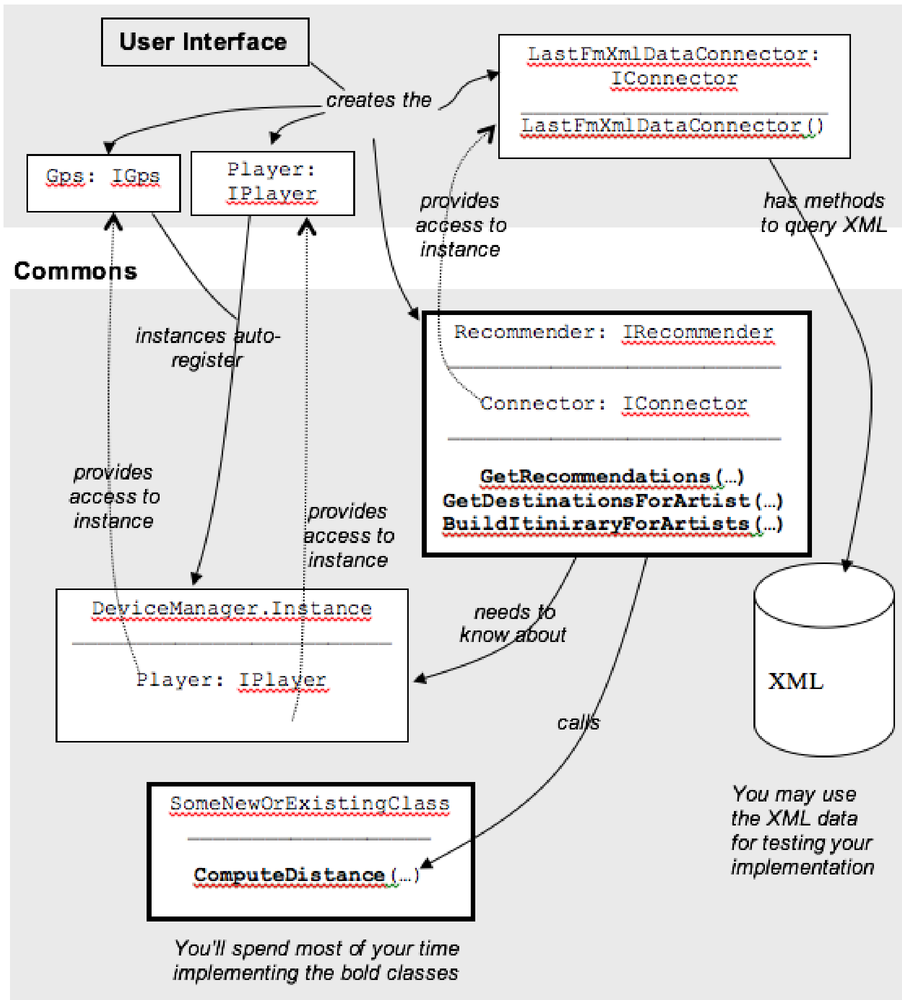


Figure C.10: General architecture diagram for the task used in the study.