Effectiveness of TDD on Unit Testing Practice

University of Oulu
Department of Information Processing
Science
Master’s Thesis
Muzamil Ahmed
Date: 15 January 2015
Abstract

**Background:** Several studies report that test driven development (TDD) has effects on the software product e.g. code quality and developers’ productivity. In recent literature reviews, the impact of TDD on source code quality is seen as a more focused area in empirical research compared to unit testing. However, the quality of production code is tightly coupled with the quality of test code.

**Objective:** The aim of this study is to investigate the impact of test driven development approach compared to an incremental test last development (ITLD), on unit test case quality. The impact is measured in terms of code coverage and mutation score metrics. The hypotheses test the differences in the quality of test cases produced using TDD and ITLD approaches.

**Method:** We conducted an experiment in an industrial setting with 24 professionals in five consecutive days. Three programing tasks, i.e. one task using ITLD and two tasks using TDD, are selected for the experiment. We extracted unit test case quality attributes i.e. mutation score and code coverage from the data collected on each day of the experiment. For the code coverage, we used the metrics; instruction, branch, method, cyclomatic complexity and line coverage. The difference of mutation score and code coverage metrics are then evaluated using non-parametric significance tests.

**Results:** The results indicate that except three metrics i.e. branch coverage, method coverage, mutation score, we could not find significant differences in terms of unit test case quality between the treatments. Subjects wrote test cases that cover more branches during TDD practice on a green-field (as a new development) task, compared to ITLD and TDD practice on a brown-field (modifications on an existing code base) task. In terms of method coverage, test cases produced using ITLD covered more methods than both TDD tasks; whereas no significant difference is found for method coverage between the both TDD tasks. In terms of mutation score, test cases written during TDD practice on a green-field task have more defect detection abilities than test cases written during ITLD practice and TDD practice on a brown-field task.

**Conclusion:** Our finding are different from previous studies performed at academic settings. We believe that other factors (i.e. task’s complexity, experiment duration and participant’s interest towards tasks) could influence the results. Therefore, future studies could be designed by minimizing the impact of these factors to get more generalizable results.

**Keywords**

Test Driven Development, Unit Test Case Quality, Experimentation, Industry Experiment
Abstract........................................................................................................................................2
1. Introduction ....................................................................................................................................5
  1.1 Problem statement .......................................................................................................................5
  1.2 Research objectives ....................................................................................................................6
  1.3 Research method .........................................................................................................................6
  1.4 Contribution ...............................................................................................................................6
  1.5 Structure .......................................................................................................................................7
2. Background and related work ........................................................................................................8
  2.1 Software testing ...........................................................................................................................9
  2.2 Unit test .......................................................................................................................................9
  2.3 Test quality attributes ..................................................................................................................9
    2.3.1 Test smells ..............................................................................................................................10
    2.3.2 Code coverage .........................................................................................................................10
    2.3.3 Mutation testing ....................................................................................................................10
  2.4 Effect of TDD on Unit Test Quality ............................................................................................12
3. Experimental design ....................................................................................................................15
  3.1 Goal ..............................................................................................................................................15
  3.2 Independent variables ................................................................................................................15
  3.3 Dependent variables ..................................................................................................................15
    3.3.1 Metrics for code coverage ......................................................................................................15
    3.3.2 Mutation score ......................................................................................................................16
  3.4 Research questions .....................................................................................................................16
  3.5 Hypotheses for R.Q.1 ................................................................................................................17
    3.5.1 Sub hypotheses for R.Q.1 ......................................................................................................17
  3.6 Hypotheses for R.Q.2 ................................................................................................................18
    3.6.1 Sub hypotheses for R.Q.2 ......................................................................................................18
  3.7 Design .........................................................................................................................................20
  3.8 Subjects .........................................................................................................................................20
  3.9 Objects .........................................................................................................................................20
    3.9.1 Green-field tasks ...................................................................................................................20
    3.9.2 Brown-field task ...................................................................................................................21
  3.10 Instrumentation ........................................................................................................................21
    3.10.1 EclEmma ..............................................................................................................................22
    3.10.2 Judy ......................................................................................................................................22
  3.11 Data collection procedure ........................................................................................................22
  3.12 Analysis procedure ...................................................................................................................22
    3.12.1 Descriptive statistics ..........................................................................................................22
    3.12.2 Graphical representation ....................................................................................................23
    3.12.3 Statistical tests .....................................................................................................................23
4. Execution .........................................................................................................................................25
  4.1 Sample .........................................................................................................................................25
4.2 Data collection performed.................................................................26

5. Results ........................................................................................................27
  5.1 Data set reduction..................................................................................27
  5.2 Descriptive statistics of code coverage ..............................................28
  5.3 Descriptive statistics of mutation score .............................................30
  5.4 Box plots ..............................................................................................31
    5.4.1 Box plot analysis of code coverage............................................31
    5.4.2 Box plot analysis of mutation score ..........................................34
  5.5 Hypothesis testing ..............................................................................36
    5.5.1 Comparison of ITLD and TDD....................................................36

6. Comparison between TDD tasks ................................................................39
  6.1 Descriptive statistics: ..........................................................................39
  6.2 Hypothesis testing ..............................................................................40
  6.3 Mann-Whitney U test ..........................................................................40
  6.4 Wilcoxon rank sum test ......................................................................40

7. Interpretation ............................................................................................42
  7.1 Evaluation and implications of results ..............................................42
  7.2 Limitations of study ............................................................................43
  7.3 Inferences ..............................................................................................44

8. Conclusions and future work .....................................................................45

9. Acknowledgements ....................................................................................46

10. References ..................................................................................................47

Appendices:......................................................................................................50
  Appendix A1: Task description – BowlingScoreKeeper..........................50
  Appendix A2. Task description - MarsRoverAPI......................................53
  Appendix A3. Task description – MusicPhone.........................................54
  Appendix A4. MusicPhone project layout.............................................55
1. Introduction

Test Driven Development (TDD) was introduced as a software development practice in early 1960s NASA’s Mercury project (Bhat & Nagappan, 2006). For more than a decade, empirical studies have been conducted to examine effects of TDD on software product (Canfora, Garcia, Piattini, & Visaggio, 2006; Damm & Lundberg, 2006; Damm & Lundberg, 2007). Causevic, Sundmark, and Punnekkat (2011) performed a literature review to identify factors, which could change when TDD is applied. The literature review of Causevic et al., (2011) highlights 18 areas including the most commonly studied factors, such as development time and code quality. A literature review by Kollanus (2010) reports that improvements in quality and productivity are the most commonly stated benefits of TDD.

In TDD, testing becomes a design activity, which helps programmer to understand the functionality of a piece of code. This development activity is mainly driven by test cases and higher quality in test cases could also lead to a higher quality software product (Madeyski, 2010). Hence, the first impact of TDD compared to a test last development is expected to be on unit test case related metrics (Madeyski, 2010).

According to Turhan, Layman, Diep, Erdogmus, and Shull (2010), testing should be improved in TDD with a growing number of automated test cases in an evolving system. Furthermore, quality of these test cases should be higher since they are written at a finer granularity level. Authors also claim that pilot studies in controlled experiments support this argument, although there is insufficient evidence to reach a conclusion from industry.

1.1 Problem statement

According to Causevic, Punnekkat, and Sundmark (2012a), researchers have mostly studied the effects of TDD on the quality of the produced code. The quality of testing, on the other hand, has found less interest in empirical research on TDD.

Some researchers believe that the quality of code is coupled with the quality of test code in both the test driven development and test last coding practices. (Madeyski, 2010) Considering the mentioned relation, there exist very few studies investigating the effect of TDD on unit test code in comparison to test last approach. Moreover, those few existing studies (Causevic et al., 2012a; Causevic, Sundmark, & Punnekkat, 2012b; Madeyski, 2010) conducted experiments in academic settings. Authors of these studies concluded that there is a need to investigate the effects of TDD on unit test case quality in industrial settings to generalize the results.
1.2 Research objectives

The goal of this study is to understand the effects of TDD on unit test case quality. Hence, we formulate our research objective as the guidelines provided by Wohlin et al., (2000).

In this research, we

Analyze [TDD]

For the purpose of [evaluation]

Compared with [Incremental Test Last]

With respect to [Unit test case quality]

From the point of view of [professionals in industry].

To accomplish our goal, we compare the incremental test last and test driven development approaches with respect to unit test case quality. For this purpose, unit test case quality is measured by two test case quality attributes i.e. code coverage and mutation score, in an industrial setting.

1.3 Research method

We follow a quantitative research approach to figure out cause-effect relationship between the variables of this study. We choose experimentation as our research methodology; which is part of empirical strategies used in software engineering (Wohlin et al., 2000). Experiments are usually performed in a laboratory environment, which provides control. The objective of an experiment is to find one or more variables and to manipulate, while other variables are controlled at a fixed level. The effects of this manipulation are measured which later can be used to perform statistical analysis (Wohlin et al., 2000).

We perform an experiment in an industrial setting with professionals. The experiment is classified as Block subject-object study (Wohlin et al., 2000). The design of our experiment is, one factor two treatments. The assignments of treatments to the subjects is based on repeated measure design. We quantify unit test case quality in terms of six metrics, and apply statistical tests on data collected from professionals in order to evaluate our hypotheses.

1.4 Contribution

This study fills the gap of industrial empirical studies on the subject under investigation. The previous studies by Causevic et al., (2012a); Causevic et al., (2012b); Madeyski, (2010) whose focus is stated in section 1.2, were conducted in academic settings. These studies highlighted the need of empirical investigations on unit test case quality as an effect of TDD, in industrial settings. This study fills the gap by conducting an experiment in an industrial setting with 24 software professionals.
Previous studies by Causevic et al., (2012a); Causevic et al., (2012b); Madeyski, (2010) used only branch coverage metric to measure code coverage. In this study we use five different metrics to measure code coverage i.e. instruction coverage, method coverage, branch coverage, cyclomatic complexity coverage and statement coverage. The usage of these five metrics helps us to avoid Mono-Method Bias validity threat.

The studies conducted by Causevic et al., (2012a) and Causevic et al., (2012b); did not use statistical tests to validate their hypotheses. This study fills this gap as we validate our hypotheses using two statistical tests i.e. Mann-Whitney U test and Wilcoxon rank sum test on multiple granularity levels of the production code.

1.5 Structure

This thesis is structured according to experiment reporting guidelines provided by Jedlitschka and Ciolkowski (2008) with slight variations. In the following section (Section 2), we present the literature about TDD, basic knowledge of testing and previous studies related to the topic. Then, we present the details of experimental design, including our goal, variables, objects and subjects of the experiment (Section 3). The fourth section presents the execution details of the experiment, and it is followed by statistical analysis of the data gathered during experimentation in Section 5. We present additional analysis by comparing the results of the two TDD tasks in Section 6, followed by the interpretations and limitations of the study in Section 7. We conclude this thesis in Section 8.
2. Background and related work

In this section we elaborate, how Test Driven Development (TDD) proceeds, the steps involved in TDD and background knowledge related to TDD. Then, we provide a basic description about software testing and literature on measuring the testing process. In this section, we also provide description about different test quality attributes which can be used to evaluate unit test quality. In the end, we present summary of existing studies on the same goal.

In the traditional test last development, unit test cases are usually written at the end of implementation. Test driven development requires writing test cases for a program’s smallest unit (smallest possible programming component) before writing the source code, and perform the same action for each unit in an iterative way (Janzen & Saiedian, 2005). TDD starts with a selection of a task, and then it is followed by writing the unit tests of the task. The next step is execution of all unit test cases to check failure of only newly added test case. This step is followed then by implementing minimum code for the task and again run all unit test cases to check whether all test cases are passed. And, if required, developers should perform refactoring at the end (Bhat & Nagappan, 2006; Turhan et al., 2010).

![TDD Flowchart](image)

**Figure 1.** Test Driven Development (Munir, Moayyed, & Petersen, 2014)

According to Figure 1, TDD is described as follows (Munir et al., 2014)

- Writing of a test case, which verifies a task or part of a selected feature.
- Execution of the test, which will fail.
- Writing of code to implement that part of feature or functionality.
- Execution of all tests.
- Correction of code, in case of any test failure.
- Re-factoring to make production code simpler, in case of all tests pass.

### 2.1 Software testing

Software testing is basically classified into two: Static and dynamic testing. In static testing, we examine the software product without executing its program through other verification and validation techniques i.e. inspections, symbolic execution (Roper 1994). Dynamic testing techniques, on the other hand, focus on examination of software via execution (Roper, 1994). These techniques are also classified as black-box and white-box (also referred as glass box).

Black-box testing is used to test the functionality of the software by providing a set of inputs and examining their output; however unawareness about the code (how much is being tested) is the major drawback of this technique. White-box testing is used to examine the construction of the software (internal working); whereas its major drawback refers to missing or unchecked functionalities.(Roper, 1994.)

Earlier testing techniques focused on defining and achieving higher level of coverage (e.g. thorough testing) to verify the program structure. Several metrics for identifying the coverage criteria were proposed, such as branch and statement coverage (Roper, 1994). Later, this trend has moved towards fault based testing techniques, also called mutation testing, which aims to test specifications and functionalities of program with more data (Roper, 1994).

### 2.2 Unit test

Hunt and Thomas (2003), describe unit test as a piece of code written by developers, with the intentions to exercise/test specific and small area of functionality of the code. Unit testing is an important process in modern software development, since this process helps to reduce bugs in the code, as well as make it easier to find problematic area of code (Aniche, Oliva, & Gerosa, 2013). According to Deursen, Moonen, Bergh, and Kok (2001), unit tests are automated and the programing language of unit tests is the same as that of production code. These tests are put under revision control and are an explicit part of the code.

### 2.3 Test quality attributes

Measuring quality is always an arguable thing for experts, as defining quality and its measure varies. Kaczanowski (2012) discusses test quality and a set of metrics representing test quality in terms of three characteristics i.e. test smell, code coverage and mutation score.
2.3.1 Test smells

Test smell is similar to the term “source code smell” (when something does not look right in production code), but it is not as standardized as source code smells (Kaczanowski, 2012). We can use static analysis tools (e.g. PMD and Findbugs) for test code as well, but test code is much simpler than production code, which makes it difficult to find smells in test code. According to Deursen et al., (2001), test code smells are different than production code smells. The difference of test smell depends on how test cases are implemented, how they interact with each other and how test cases are organized. Deursen et al., (2001) described a few test smell: mystery guest, resource optimization, test run war, general fixture, eager test, lazy test, assertion roulette, indirect testing, for testers only, sensitive equality and test code duplication. Kczanowski (2012) recommends eliminate smells in both test and source code by following best programming practices.

According to the mentioned definitions, test smells can be categorized as static testing because there is no need to execute code to detect test smells in the software product. As described in section 2.1 (software testing), in this study we examine unit test quality using dynamic testing techniques (through execution of tests on the source code). Furthermore, Kczanowski (2012) argues that research on test smells are not as much mature as research on code smell, different authors define it differently and there is no specialized tool that aim to find test smells only. Due to these reasons, we decided not to include test smell metrics to evaluate unit test case quality.

2.3.2 Code coverage

Testing is useless if we fail to execute a faulty element in the code (Pezze & young 2008). For example, if we have a fault in 12th statement of the source code, this fault could only be revealed by test cases that exercise this statement.

To measure the quality of test cases, code coverage is a commonly used technique. It measures which part (e.g. line, statements, and branches) of the production code is exercised/executed during test execution. There are different ways to measure code coverage. A literature review by Shahid, Ibrahim, and Naz (2011), gives 12 different metrics to measure code coverage. In this experiment, we choose five of those (described in detail in section 3.3); branch coverage is selected because it is previously used in Causevic et al., (2012a), Causevic et al., (2012b) and Madeyski (2010), the rest are chosen based on the tool's capabilities. Code coverage results are mentioned in terms of percentages, which can vary from 0 to 100%. If we cannot achieve 100% code coverage it means we are missing some code pieces to test and more test cases are required to achieve 100% coverage. (Shahid et al., 2011.)

2.3.3 Mutation testing

Mutation testing is a way to measure the ability of a test case to catch defects in the code. To verify how good our test is, we create program mutants that are multiple versions of the program code in which different types of defects are injected.
According to Pezzè and Young (2008), a mutant is a program which is different from the original program at one syntactic item. A mutation testing tool creates mutant of a program by applying a mutation operator at a single location of the program (Frankl, Weiss, & Hu, 1997). Support for mutation operators is considered as an important characteristic for a mutation testing tool (Madeyski & Radyk, 2010). Mutation operator is typically formed when one variable is replaced by another and one relation operator is replaced by another (Frankl et al., 1997). Madeyski and Radyk (2010), uses 16 mutation operators in their tool named, “Judy” to generate mutants; Table 1 explains all 16 operators used by Judy mutation tool. After executing a test case, if a mutant can be distinguished from the original program, it is said that test case kills the mutant. If the mutant cannot be distinguished, it is called as an equivalent mutant (Frankl et al., 1997). Mutation testing is used to check the fault detection effectiveness of a given test suite. Mutation score can be represented as the percentage of faults detected by the test suite. (Madeyski & Radyk, 2010.) A higher mutation score shows higher effectiveness of the test suite in detecting the (injected) defects (Frankl et al., 1997). In order to compute mutation score of a program, all test cases in that program’s test suite should be passed before execution of a mutation tool.

**Table 1. Mutation operators supported by Judy**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Example Mutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Absolute value insertion</td>
<td>a = 4<em>x;→ a = 4</em>xyz(x);</td>
</tr>
<tr>
<td>AOR</td>
<td>Arithmetic operator replacement</td>
<td>a = x*y;→ a = x - y;</td>
</tr>
<tr>
<td>LCR</td>
<td>Logical connector replacement</td>
<td>a = x</td>
</tr>
<tr>
<td>ROR</td>
<td>Relational operator replacement</td>
<td>if (x &lt; y)→ if(x &gt; y)</td>
</tr>
<tr>
<td>UOI</td>
<td>Unary operator insertion</td>
<td>a = 4<em>x;→ a= 4</em>-x;</td>
</tr>
<tr>
<td>UOD</td>
<td>Unary operator deletion</td>
<td>if (x &lt; -y)→ if (x &lt; y)</td>
</tr>
<tr>
<td>SOR</td>
<td>Shift operator replacement</td>
<td>if (x &lt;&lt; y)→ if (x &gt;&gt; y)</td>
</tr>
<tr>
<td>LOR</td>
<td>Logical operator replacement</td>
<td>if (x &amp; y)→ if (x</td>
</tr>
<tr>
<td>COR</td>
<td>Conditional operator replacement</td>
<td>if (x&amp;&amp;y)→ if (x&amp;y)</td>
</tr>
<tr>
<td>ASR</td>
<td>Assignment operator replacement</td>
<td>a+ = 4;→ a- = 4;</td>
</tr>
<tr>
<td>EOA</td>
<td>Reference assignment and content assignment replacement</td>
<td>List x1; x2; x1 = new List(); x1 = x2;→ x1 = x2:clone()</td>
</tr>
<tr>
<td>EOC</td>
<td>Reference comparison and content comparison replacement</td>
<td>Integer a = new Integer(1); Integer b = new Integer(1); boolean x = (a == b);→ boolean x = (a.equals(b));</td>
</tr>
<tr>
<td>JTD</td>
<td>this keyword deletion</td>
<td>this.x = x;→ x = x;</td>
</tr>
<tr>
<td>JTI</td>
<td>this keyword insertion</td>
<td>this.x = x;→ this.x = this.x;</td>
</tr>
<tr>
<td>EAM</td>
<td>Accessor method change</td>
<td>point.getX();→ point.getY ();</td>
</tr>
<tr>
<td>EMM</td>
<td>Modifier method change</td>
<td>point.setX(1);→ point.setY (1);</td>
</tr>
</tbody>
</table>
2.4 Effect of TDD on Unit Test Quality

Causevic, et al., (2011) performed a systematic review on TDD. They filtered 48 studies investigating the effects of TDD on several factors. From these studies, Causevic et al., (2011) categorize 18 factors, which hinder the adaptation of TDD. The goals of most of these 48 studies are to investigate the effects of TDD on external code quality, internal code quality and performance improvement. Only one study (Madeyski, 2010) is found with a focus of unit test case quality during a TDD application. Madeyski (2010) performed an experiment with master students in an academic setting. They used branch coverage and mutation score to evaluate the effects of TDD on unit test case quality, compared to unit test case quality measured during a test last approach. Their experiments could not reveal any significant differences in terms of branch coverage and mutation score of unit test cases between TDD and test last approach. Causevic et al., (2012a) and Causevic et al., (2012b) later extended Madeyski (2010) experiment with one experiment and report it in two different studies, conducted in academia with master students. They found no significant difference in branch coverage and mutation score of unit test case produced using test first and test last approaches. The details of the experiments conducted by Causevic et al., (2012a), Causevic et al., (2012b) and Madeyski (2010) are summarized in Table 2.

A recent systematic literature review conducted by Munir et al., (2014), identifies patterns and themes of the studies on TDD. They extracted eight categories from studies that focus on the impact of TDD over test last development. These categories are: 1. Productivity, 2. External quality, 3. Developer opinion, 4. Internal code quality, 5. Effort/time, 6. Conformance, 7. Size and 8. Robustness. Munir et al., (2014) did not mention any category related to unit test case quality, or any study on the impact of TDD on unit test cases quality. However, authors listed branch coverage and cyclomatic complexity metrics under the internal code quality category, as these metrics were previously used by researchers to evaluate this category. Furthermore, some artifacts of unit test cases i.e. number of test case passed, total number of assertions passed/failed and total number of test case written are discussed under the size and external quality categories (Munir et al., 2014).
<table>
<thead>
<tr>
<th>Title</th>
<th>(Madeyski, 2010)</th>
<th>(Causevic et al., 2012b)</th>
<th>(Causevic et al., 2012a)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>Test-First (TF) vs. Test-Last (TL) programming practices are examined with regard to branch coverage and mutation score indicator of unit tests.</td>
<td>Compare unit test case effectiveness when developing software using test driven development (test-first) and traditional (test-last) approaches.</td>
<td>Compare unit test case effectiveness when developing software using test driven development (test-first) and traditional (test-last) approaches.</td>
</tr>
<tr>
<td><strong>Hypotheses</strong></td>
<td>1. There is no difference in branch coverage (BC) between the TF and the TL projects.</td>
<td>There is no significant difference between the qualities of the test artefacts produced by test-first or test-last developers.</td>
<td>There is no significant difference between the qualities of the test artefacts produced by test-first or test-last developers.</td>
</tr>
<tr>
<td></td>
<td>2. There is no difference in mutation score indicator (MSI) between the TF and the TL projects.</td>
<td>There is no significant difference between the qualities of the test artefacts produced by test-first or test-last developers.</td>
<td>There is no significant difference between the qualities of the test artefacts produced by test-first or test-last developers.</td>
</tr>
<tr>
<td><strong>Independent Variables</strong></td>
<td>Software development technique – TF or TL.</td>
<td>Software development technique – TF or TL.</td>
<td>Software development technique – TF or TL.</td>
</tr>
<tr>
<td><strong>Dependent Variables</strong></td>
<td>Branch coverage &amp; Mutation score indicator</td>
<td>Total number of failing assertions &amp; Code Coverage &amp; Mutation Score</td>
<td>Defect Detecting Ability &amp; Code Coverage &amp; Mutation Score</td>
</tr>
<tr>
<td><strong>Methodology</strong></td>
<td>Controlled experiment</td>
<td>Controlled experiment</td>
<td>Controlled experiment</td>
</tr>
<tr>
<td><strong>Design</strong></td>
<td>One Factor Two Treatments</td>
<td>One Factor Two Treatments</td>
<td>One Factor Two Treatments</td>
</tr>
<tr>
<td><strong>Subjects</strong></td>
<td>Third and fourth-year graduate MSc software engineering students enrolled in EBusiness Technologies (EBT) course</td>
<td>Fourteen (software engineering master) students enrolled in the Software Verification and Validation course at Mälardalen University</td>
<td>Fourteen (software engineering master) students enrolled in the Software Verification and Validation course at Mälardalen University</td>
</tr>
<tr>
<td><strong>Tools</strong></td>
<td>Clover: To collect Branch Coverage score. Judy: To collect Mutation score. Activity Sensor: To measure the development</td>
<td>EclEmma: To collect Branch Coverage score. Judy: To collect Mutation score. SVN tool: To collect development log data.</td>
<td>EclEmma: To collect Branch Coverage score. Judy: To collect Mutation score. SVN tool: To collect development log data.</td>
</tr>
</tbody>
</table>

**Table 2. Summary of related work**
<table>
<thead>
<tr>
<th>Instrumentation</th>
<th>Requirement artefacts (user stories). Instructions with examples.</th>
<th>Instructions with examples.</th>
<th>Instructions with examples.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data collection procedure</td>
<td>Pre-test and post-test questionnaires. The measurement data were collected automatically by tools described above.</td>
<td>Source code and log obtained from SVN. Student filled a survey questionnaire.</td>
<td>Source code and log obtained from SVN. Student filled a survey questionnaire.</td>
</tr>
<tr>
<td>Analysis procedure</td>
<td>The experiment data are analyzed with descriptive analysis and statistical tests.</td>
<td>The results produced from both treatments were compared with each other in tables.</td>
<td>The results produced from both treatments were compared with each other in tables.</td>
</tr>
<tr>
<td>Limitations (mentioned in the articles)</td>
<td>Students as subjects. Small scale objects of investigation. Short duration of the experiment.</td>
<td>Students as subjects. Small scale objects of investigation. Short duration of the experiment.</td>
<td>Students as subjects. Small scale objects of investigation. Short duration of the experiment.</td>
</tr>
<tr>
<td>Results</td>
<td>No significance difference found.</td>
<td>No significance difference found.</td>
<td>No significance difference found. Test first participants has more positive test cases.</td>
</tr>
</tbody>
</table>

The major focus of studies, listed in Table 2, is on unit test case quality. These studies were conducted with graduate (MSc level) students, and authors used branch coverage and mutation score as two variables to measure unit test case quality. Our study is an extension of those three studies. We conduct an industrial experiment with professionals located in three different sites of a company. We extend the list of dependent variables to measure unit test case quality. We choose two objects with different size and levels of complexity for TDD experimentation and evaluate the progress of subjects in TDD practice and its effect on unit test case quality as an additional analysis of this study. We compare the results of incremental test last development (ITLD) object with two TDD objects separately to see the effect on unit test case quality using the two development approaches. We evaluate our hypotheses for code coverage metrics, with Mann-Whitney U test on class level and Wilcoxon rank sum test on project level granularity of the production code. We evaluate our hypothesis for mutation score with Wilcoxon rank sum test. Our experiment also shares a common goal with the prior studies (Causevic et al., 2012a; Causevic et al., 2012b; Madeyski, 2010), as we compare unit test case effectiveness implemented using test driven development and test last development approaches. (Causevic et al., 2012a; Causevic et al., 2012b) used EclEmma tool for code coverage analysis in their experiments. For mutation score Judy has been used by Cauvevic et al., (2012a), Causevic et al., (2012b) and Madeyski (2010). Our study is an extension of these studies therefore; we use the same tools to keep uniformity of results.
3. Experimental design

This section provides description about the outcomes of the experiment planning phase. In this section, we present the goal, research questions and variables of our experiment. Moreover, we describe the selected metrics, and formulate our hypotheses and sub hypotheses.

3.1 Goal

The goal of this study is to evaluate the effectiveness of TDD approach on unit test case quality. In order to achieve this goal, we answer two main research questions in this study. Our first research question is “What is the effect of TDD on the quality of unit test cases compared to an incremental test last approach (ITL)?”. We perform an additional analysis of the experiment to investigate the learning/practice effect of TDD on unit test case quality; this is serving as second research question of the study.

3.2 Independent variables

The independent variable of our study is a development practice i.e. Incremental Test Last Development (referred as ITLD) and Test Driven Development (referred as TDD). Subjects implement one task using ITLD and two tasks using TDD, referred as TDD1 and TDD2 respectively. Incremental test last is similar with traditional test last approach, as the specifications are divided into small user stories in both the approaches. The difference between ITLD and traditional test last development practice is, in ITLD, participants have to implement and test each user story in an increment and then move to the next user story as a next increment.

3.3 Dependent variables

The dependent variable of our study is unit test case quality which is measured by two quality attributes that are code coverage and mutation score.

3.3.1 Metrics for code coverage

We use the following metrics to calculate code coverage in this study.

Instruction Coverage

Single Java byte code instructions are the smallest unit in coverage counters. Instruction coverage gives execution information about amount of code execution. It is not
dependent on source formatting and available even if debug information is missing in class files. (“JaCoCo - Coverage Counter,” n.d.)

**Line coverage**

Line coverage also known as statement and basic block coverage (Chen, Xu, Yang, & Chen, 2003) is simple to measure, if a line is executed during test execution then this line is covered (Kaczanowski, 2012). Line coverage is different from instruction coverage as in instruction coverage we compute coverage of java byte code.

**Branch coverage**

Decision points are main focus area of branch coverage, it focuses on conditional statements with logical operators (i.e. && or ||) (Kaczanowski, 2012). Complete statement coverage can be achieved without exercising all possible branches (Pezzè & Young, 2008). For example, if we do not have any statement in false branch of code then we can achieve 100% statement coverage without executing false branch of code.

**Cyclomatic complexity**

Sometimes, a fault is exposed by exercising some sequences of conditions (e.g. particular paths) (Pezzè & Young, 2008). The cyclomatic complexity is closely related with the amount of work required to test a software product. If a program’s cyclomatic complexity is lesser than the number of test paths, it means that there is a less test coverage. In other words more tests should be developed. (Mccabe, 1976.)

**Methods coverage**

Method coverage refers to the execution of non-abstract methods (contains at least one instruction) during testing (“JaCoCo - Coverage Counter,” n.d.).

**3.3.2 Mutation score**

We use mutation score to measure quality of test suite quantitatively. It is calculated as, total number of killed mutants over total number of non-equivalent mutants ratio (Madeyski & Radyk, 2010).

**3.4 Research questions**

In order to achieve the objectives of this study we formulate the following research questions.
Research Question 1: What is the effect of TDD on the quality of unit test cases compared to an incremental test last approach (ITL)?

We have further defined two sub research questions based on two attributes used to measure unit test case quality.

R.Q 1.1: What is the effect of TDD on the code coverage compared to ITL?

R.Q 1.2: What is the effect of TDD on the mutation score compared to ITL?

We also study the learning effect (practice) of TDD on unit test case quality, as an additional analysis of the study. To accomplish that, we form RQ2 and compare the differences between the unit test case qualities of two TDD tasks.

Research Question 2: How does the practice on TDD affect unit test case quality?

3.5 Hypotheses for R.Q.1

Based on Research Question 1, we first define general null and alternative hypotheses.

- \( H_{01} \mu(TCQ)_{TDD} = \mu(TCQ)_{ITLD} \) – There is no difference of unit test case quality (TCQ) between the TDD and the ITLD projects.
- \( H_{11} \mu(TCQ)_{TDD} \neq \mu(TCQ)_{ITLD} \) – There is significant difference of unit test case quality (TCQ) between the TDD and the ITLD projects.

3.5.1 Sub hypotheses for R.Q.1

As described in dependent variable section, we evaluate unit test quality with multiple matrices, so we formulate multiple sub null hypotheses and alternative hypotheses for these metrics.

- There is no difference in branch coverage (BRCov) between the TDD and the ITLD projects.
  \( H_{01.1} \mu(BRCov)_{TDD} = \mu(BRCov)_{ITLD} \)

- There is significant difference in branch coverage (BRCov) between the TDD and the ITLD projects.
  \( H_{11.1} \mu(BRCov)_{TDD} \neq \mu(BRCov)_{ITLD} \)

- There is no difference in instruction coverage (INSCov) between the TDD and the ITLD projects.
  \( H_{01.2} \mu(INSCov)_{TDD} = \mu(INSCov)_{ITLD} \)

- There is significant difference in instruction coverage (INSCov) between the TDD and the ITLD projects.
  \( H_{11.2} \mu(INSCov)_{TDD} \neq \mu(INSCov)_{ITLD} \)
• There is no difference in method coverage (MTCov) between the TDD and the ITLD projects.
  \[ H_{01.3} \mu(\text{MTCov})_{TDD} = \mu(\text{MTCov})_{ITLD} \]

• There is significant difference in method coverage (MTCov) between the TDD and the ITLD projects.
  \[ H_{11.3} \mu(\text{MTCov})_{TDD} \neq \mu(\text{MTCov})_{ITLD} \]

• There is no difference in line coverage (LOCov) between the TDD and the ITLD projects.
  \[ H_{01.4} \mu(\text{LOCov})_{TDD} = \mu(\text{LOCov})_{ITLD} \]

• There is significant difference in line coverage (LOCov) between the TDD and the ITLD projects.
  \[ H_{11.4} \mu(\text{LOCov})_{TDD} \neq \mu(\text{LOCov})_{ITLD} \]

• There is no difference in cyclomatic complexity coverage (COMCov) between the TDD and the ITLD projects.
  \[ H_{01.5} \mu(\text{COMCov})_{TDD} = \mu(\text{COMCov})_{ITLD} \]

• There is significant difference in cyclomatic complexity coverage (COMCov) between the TDD and the ITLD projects.
  \[ H_{11.5} \mu(\text{COMCov})_{TDD} \neq \mu(\text{COMCov})_{ITLD} \]

• There is no difference in mutation score (MUTScore) between the TDD and the ITLD projects.
  \[ H_{01.6} \mu(\text{MUTScore})_{TDD} = \mu(\text{MUTScore})_{ITLD} \]

• There is significant difference in mutation score (MUTScore) between the TDD and the ITLD projects.
  \[ H_{11.6} \mu(\text{MUTScore})_{TDD} \neq \mu(\text{MUTScore})_{ITLD} \]

3.6 Hypotheses for R.Q.2

To evaluate second question of study, we verify following hypotheses.

• \( H_{02.1} \mu(\text{TCQ})_{TDD1} = \mu(\text{TCQ})_{TDD2} \) – There is no difference in unit test case quality (TCQ) between the TDD1 and the TDD2 projects.

• \( H_{12.2} \mu(\text{TCQ})_{TDD1} \neq \mu(\text{TCQ})_{TDD2} \) – There is significant difference of unit test case quality (TCQ) between the TDD1 and the TDD2 projects.

3.6.1 Sub hypotheses for R.Q.2

• There is no difference in branch coverage (BRCov) between the TDD1 and the TDD2 projects.
  \[ H_{02.3} \mu(\text{BRCov})_{TDD1} = \mu(\text{BRCov})_{TDD2} \]
• There is significant difference in branch coverage (BRCov) between the TDD1 and the TDD2 projects.
  
  \[ H_{12.1} \mu(BRCov)_{TDD1} \neq \mu(BRCov)_{TDD2} \]

• There is no difference in instruction coverage (INSCov) between the TDD1 and the TDD2 projects.
  
  \[ H_{02.2} \mu(INSCov)_{TDD} = \mu(INSCov)_{TDD} \]

• There is significant difference in instruction coverage (INSCov) between the TDD1 and the TDD2 projects.
  
  \[ H_{12.2} \mu(INSCov)_{TDD1} \neq \mu(INSCov)_{TDD2} \]

• There is no difference in method coverage (MTCov) between the TDD1 and the TDD2 projects.
  
  \[ H_{02.3} \mu(MTCov)_{TDD1} = \mu(MTCov)_{TDD2} \]

• There is significant difference in method coverage (MTCov) between the TDD1 and the TDD2 projects.
  
  \[ H_{12.3} \mu(MTCov)_{TDD1} \neq \mu(MTCov)_{TDD2} \]

• There is no difference in line coverage (LOCov) between the TDD1 and the TDD2 projects.
  
  \[ H_{02.4} \mu(LOCov)_{TDD1} = \mu(LOCov)_{TDD2} \]

• There is significant difference in line coverage (LOCov) between the TDD1 and the TDD2 projects.
  
  \[ H_{12.4} \mu(LOCov)_{TDD1} \neq \mu(LOCov)_{TDD2} \]

• There is no difference in cyclomatic complexity coverage (COMCov) between the TDD1 and the TDD2 projects.
  
  \[ H_{02.5} \mu(COMCov)_{TDD1} = \mu(COMCov)_{TDD2} \]

• There is significant difference in cyclomatic complexity coverage (COMCov) between the TDD1 and the TDD2 projects.
  
  \[ H_{12.5} \mu(COMCov)_{TDD1} \neq \mu(COMCov)_{TDD2} \]

• There is no difference in mutation score (MUTScore) between the TDD1 and the TDD2 projects.
  
  \[ H_{02.6} \mu(MUTScore)_{TDD1} = \mu(MUTScore)_{TDD2} \]

• There is significant difference in mutation score (MUTScore) between the TDD1 and the TDD2 projects.
  
  \[ H_{12.6} \mu(MUTScore)_{TDD1} \neq \mu(MUTScore)_{TDD2} \]
3.7 Design

We choose one factor i.e. software development method with two treatments (Incremental Test Last Development and Test Driven Development), as design of the experiment. We follow repeated measure design in our experiment. As per definition of repeated measure design, the subjects of the experiment were exposed to both treatments, and we measured their performance on three experimental days.

3.8 Subjects

We conduct experiment in three different sites of the same company; Helsinki (Finland), Oulu (Finland) and Kuala Lumpur (Malaysia). The company operates at multinational level and provides security services and products to protect digital life of consumers and business for over a couple of decades.

We applied the convenience sampling – a non-probability sampling technique. In total 24 participants volunteered for the experiment, on the chosen sites of company. More specifically, six participants attended the experiment in Helsinki (Finland), 11 attended in Oulu (Finland) and seven attended in Kuala Lumpur (Malaysia).

3.9 Objects

Three programing tasks are provided to participants as the objects of the experiment. Out of the three programing tasks, two tasks can be classified as green field tasks, whereas one can be referred as a brown field task. Green field tasks are implemented from scratch (as a new development), whereas modification or additions are made to the existing code base in brown field tasks.

3.9.1 Green-field tasks

In this section we provide description about green-field tasks implemented in the experiment.

Bowling Score Keeper

The main goal of this application is to calculate score of a bowling game. There is no graphical interface required for this game, participants have to work with class objects (object-oriented design) and Junit test cases to give input and check the output. The complete explanation of the task is available in appendix A1.
MarsRoverAPI

Participants have to implement an API to move Rover on a grid. Initial starting point and direction (N.E.W.S) the rover is facing are provided. Rover move left, right, forward and backward based on character commands provided. Rover is able to detect and report obstacles. The order of user stories explained in the specification’s document also depicts the difficulty level of the functionality associated with the user stories. The complete explanation is available in appendix A2.

3.9.2 Brown-field task

In this section we provide description about brown-field task implemented in the experiment.

MusicPhone

Music Phone application recommends artist to users, based on their likeness. It helps users to find upcoming concerts’ events from the LastFM website data. This application works on MP3 capable and GPS enabled mobile phones. In this task, applicants have to implement some requirements on existing code base. Developers have to create functionalities of a GPS based mobile phone application. The requirement is to perform five assignments. The description not just notifies the names of the methods, but also how they relate to the design of the current application architecture. A set of smoke tests is also part of the description of the intended assignment artefact, in order to make the applicants comprehend the structure of the application. Detailed description about MusicPhone existing architecture and set of assignments can be found in appendix A4 and A3.

3.10 Instrumentation

To attain optimal results from the experiment, training sessions were arranged at sites of experiment. The focus of training sessions was to introduce subject to TDD, unit testing practices and how to develop an application using these practices. Java is selected as the programming language, and the development environment is Eclipse IDE with JUnit plugin which is used for unit testing. A pre-experiment questionnaire was filled online by subjects to get more background information about their experience and skills. We used the questionnaire to see how heterogeneous our sample is. We aimed to have a group of subjects that show heterogeneity in terms of programming skills and experience. Therefore, we collected demographic information and confirmed that we achieved our objective of having as heterogeneous group as possible. Results of the questionnaire are presented in section 4.1.

Virtual machines provided to participants for experiment. Supporting software i.e. operating system, java development kit and tools required for development (i.e. eclipse) were pre-installed on these virtual machines. We use Judy and EclEmma tools to collect mutation score and code coverage results respectively.
3.10.1 EclEmma

EclEmma is a JaCoCo code coverage library based tool to calculate code coverage for Eclipse (“EclEmma - Java Code Coverage for Eclipse,” n.d.). It works on Java byte code, which means it can work without source code. It provides coverage analysis for instruction, line, branch, method, and cyclomatic complexity at class level of software product. This tool outputs results in a comma separated file format.

3.10.2 Judy

Mutation score is an effective way to measure test case effectiveness, but it is time and resource consuming activity, and it is not feasible to measure this metric without an automated tool. Support for mutation operator can be described as an important characteristic of mutation score tool. (Madeyski & Radyk, 2010.)

We used Judy (java mutation tool) to get mutation score indicators of experiment tasks. Judy support 16 predefined mutation operators (described in section 2.3.3) to achieve high mutation performance (Madeyski & Radyk, 2010). It provides mutation for software product at project level. This tool outputs results in a comma separated file format.

3.11 Data collection procedure

At the end of each experiment day, images of virtual machines are copied in USB stick. We collect production and test code from these virtual machines after completion of the experiment each day. In order to have the desired data, we execute EclEmma on code as Eclipse plugin, whereas Judy is executed via command line interface. We collect the output results from both tools in comma-separated files.

3.12 Analysis procedure

In this section, we present how we analyze our findings from the experiment.

3.12.1 Descriptive statistics

Our analysis procedure starts with comparison of descriptive statistics; we compare mean, median, standard deviation, minimum and maximum for all three tasks with respect to test quality metrics (described in section 3.3).
3.12.2 Graphical representation

We evaluate our results graphically with box plot. In Figure 2, we present a sample boxplot described by (Wickham & Stryjewski, 2011). The box plot consists on following component to provide robust summary of dataset distribution.

- Median
- Upper and lower fourths also called quartiles (define range of distribution)
- Lines (also called whiskers) define limits of distribution (either min of max or sometime 1.5 of interquartile range to exclude extreme outliers).
- Outliers: individual points showing extreme values. (Potter, Hagen, Kerren, & Dannenmann, 2006.)

![Box-plot diagram](image)

*Figure 2. Box-plot (Wickham & Stryjewski, 2011)*

3.12.3 Statistical tests

The statistical tests are run to test whether or not null hypotheses can be refuted. In this study, we evaluate our hypotheses with two statistical tests. As the first step, we evaluate our hypotheses with the help of Mann-Whitney U test on class level (raw) data, with a significance level, $p$ of 0.05. As second step, we derive second dataset by aggregating class level data into project level data. We confirm statistical results from Mann-Whitney U test with Wilcoxon rank sum test on paired project level data, with a significance level, $p$ of 0.05. As tool for mutation score “Judy” generates results only at project level data that is why we evaluate hypothesis for mutation score with Wilcoxon rank sum test only.
**Mann-Whitney U test**

We used Mann-Whitney test to verify hypotheses of our study. As our study has two treatments (TDD and ITLD), we need to compare two independent samples of data representing these treatments. Although T-test satisfies these requirements, we are not sure that our data is normally distributed. That is why, we choose a non-parametric alternative of t-test, Mann-Whitney U test (Wohlin et al., 2000). Using this test, we check if the medians of two samples representing a unit test quality metric (at class level) are the same among TDD and ITL groups.

**Wilcoxon rank sum test**

Wilcoxon rank sum test is a non-parametric alternative to paired t-test (Wohlin et al., 2000). When design of the experiment involve one factor two treatment and a paired comparison then Wilcoxon rank sum test is used. For code coverage metrics, we applied Wilcoxon rank sum test at project level data, by aggregating class level metrics into project level (also called subject level). For instance, for branch coverage metric, we calculated the mean (MEAN), median (MEDIAN), standard deviation (SD), minimum (MIN) and maximum (MAX) of branch coverage values obtained from classes in each project and created a new dataset. Then, we compare this project level data between subjects using Wilcoxon rank sum test. Judy (tool for mutation score calculation) generates results at project level, because of this we execute only Wilcoxon rank sum test for mutation score hypothesis evaluation.
4. Execution

This section, we describe how experimental plan is implemented. In this section, we present schedule of the experiment followed by sample details and data collection.

![Figure 3. Schedule of experiment](taken from fidipro project material)

In Figure 3, we present schedule of the experiment. Subjects have to implement three tasks in five days (each task on assigned day). On the first day, they got training about unit testing and were asked to implement MarsRoverAPI task using ITLD approach. On the second day, training about TDD was provided to all subjects, and they implemented Bowling Score Keeper task (which we refer as TDD1) using TDD. For the third and fourth days, subjects practiced test driven development by themselves, i.e. they were asked to practice TDD in their daily work. Finally, they were asked to implement MusicPhone task (which we refer as TDD2) using TDD.

4.1 Sample

We have selected 24 professionals from a software company (described in section 3.8). In order to see, how heterogeneous our sample is, we conducted a pre experiment online questionnaire. We collected information about subjects’ experience (in years) and professional skills in context of the study. In Table 3, we summarize demographics of subjects of the experiment.
Table 3. Demographics data (Salman, 2014)

<table>
<thead>
<tr>
<th>Experience Categories</th>
<th>Programming Experience</th>
<th>Java Programming Experience</th>
<th>Unit Testing Experience</th>
<th>JUnit Experience</th>
<th>TDD Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expert (&gt;10 years)</td>
<td>10</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Intermediate (5-&lt;=10 years)</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Novice (2-&lt;=5 years)</td>
<td>4</td>
<td>9</td>
<td>14</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>No experience (&lt;2 years)</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>Total</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
</tbody>
</table>

4.2 Data collection performed

Data collection took place according to plan (described in section 3.11) without any deviation.
5. Results

In this section, we report the results and analyze the data. In the following sub section, we explain filtration procedure of dataset. We report the results using descriptive statistics, box plots and statistical tests, as explained in the analysis procedure section.

5.1 Data set reduction

We had in total of 24 participants who attended our experiment in three experimental sites. Due to error in code (while compiling projects of three participants) and incomplete tasks (of four participants), we are not able to calculate scores. To avoid any false statistics, we removed those participants’ data from comparative analysis. Details of these errors can be found in Table 4.

**Table 4.** Data reduction for coverage report

<table>
<thead>
<tr>
<th>Participant ID</th>
<th>The reason for exclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSH02</td>
<td>Errors in Code of MarsRover (ITLD) and MusicPhone (TDD2).</td>
</tr>
<tr>
<td>FSH04</td>
<td>Does not have source code of MusicPhone (TDD).</td>
</tr>
<tr>
<td>FSH11</td>
<td>Does not have source code of MusicPhone (TDD).</td>
</tr>
<tr>
<td>FSK18</td>
<td>Errors in Code of MarsRover (ITLD) and Bowling Score Keeper (TDD1).</td>
</tr>
<tr>
<td>FSOU2803</td>
<td>Does not have test code in MarsRover (ITLD).</td>
</tr>
<tr>
<td>FSOU2806</td>
<td>Errors in Code of MarsRover (ITLD).</td>
</tr>
<tr>
<td>FSOU2807</td>
<td>Does not have test code in MarsRover (ITLD).</td>
</tr>
</tbody>
</table>

Table 5 presents initial coverage scores for all the tasks. We observe abnormal stats (zero coverage values of median) for TDD2 (Music Phone task). After digging more into it, we found that as TDD2 task (Music Phone) was provided to developers in semi-implemented form, no unit test case was written by the participants to test this legacy code, and hence, we got zero coverage values for those legacy methods. To eliminate this, we applied a filtering i.e. we removed all methods, which were not implemented or modified by the participants during experiment, and computed the metrics based on the rest of the methods. Table 6 represents the final statistics for code coverage metrics.
5.2 Descriptive statistics of code coverage

The descriptive statistics of all metrics (described in section 3.3) are summarized in Table 6, taken at project level. We have three rows for development methods (ITLD, TDD1, and TDD2). Columns represent the metrics, namely INSCov = instruction covered percentage, BRCov = branch covered percentage, LOCov = line covered percentage, COMCov = cyclomatic complexity covered percentage, MTCov= method covered percentage. In this section, we compare these metrics in terms of their mean, median and standard deviation (StdD) values.
Table 6. Final descriptive statistics for code coverage metrics after filtering on Music Phone task

<table>
<thead>
<tr>
<th>Task</th>
<th>INS Cov</th>
<th>BRCov</th>
<th>LOCov</th>
<th>COM Cov</th>
<th>MTCov</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ITLD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>71.8</td>
<td>36.91</td>
<td>75.92</td>
<td>62.8</td>
<td>81</td>
</tr>
<tr>
<td>Median</td>
<td>83.18</td>
<td>32.5</td>
<td>81.75</td>
<td>66.67</td>
<td>90</td>
</tr>
<tr>
<td>StdD</td>
<td>26.99</td>
<td>34.47</td>
<td>23.3</td>
<td>25.44</td>
<td>23.41</td>
</tr>
<tr>
<td>Minimum</td>
<td>10.11</td>
<td>0</td>
<td>16.67</td>
<td>7.69</td>
<td>11.11</td>
</tr>
<tr>
<td>Maximum</td>
<td>100</td>
<td>96.43</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td><strong>TDD-1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>76.06</td>
<td>66.48</td>
<td>72.84</td>
<td>66.24</td>
<td>65.49</td>
</tr>
<tr>
<td>Median</td>
<td>90.77</td>
<td>92.71</td>
<td>85.44</td>
<td>76.92</td>
<td>73.21</td>
</tr>
<tr>
<td>StdD</td>
<td>32.03</td>
<td>43.19</td>
<td>31.3</td>
<td>29.98</td>
<td>29.56</td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td><strong>TDD-2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>61.49</td>
<td>27.01</td>
<td>62.06</td>
<td>59.4</td>
<td>61.41</td>
</tr>
<tr>
<td>Median</td>
<td>68.26</td>
<td>0</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>StdD</td>
<td>35.84</td>
<td>41.5</td>
<td>33.66</td>
<td>32.07</td>
<td>31.01</td>
</tr>
<tr>
<td>Minimum</td>
<td>3.25</td>
<td>0</td>
<td>4.79</td>
<td>7.14</td>
<td>8.33</td>
</tr>
<tr>
<td>Maximum</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

From Table 6, we see that there are differences between the ITLD and TDD1 tasks with respect to instruction, and branch coverage. In terms of instruction coverage, subject applying TDD score higher than those applying ITLD. On average, 71.8% (median: 83.18%, and standard deviation: 26.99%) of instruction are covered in ITLD task, whereas, 76.06% of instructions (median: 90.77% and standard deviation: 32.03%) are covered in TDD1 task. In terms of branch coverage, subjects applying TDD score higher than those applying ITLD. In other words, on average, 36.9% (median: 32.5% and standard deviation: 34.4%) of branches are covered in ITLD task, while 66.4% of branches (median: 92.7%, standard deviation: 43.1%) are covered in TDD1 task.

In terms of line coverage, the differences between ITLD and TDD1 tasks are small in terms of mean and median, while the standard deviation in TDD1 is higher. In other words, on average 75.92% (median: 81.75% and standard deviation: 23.30%) of lines are covered in ITLD task, on the other hand, on average 72.84% (median: 85.44% and standard deviation: 31.30%) lines are covered in TDD1 task. Cyclomatic complexity also shows minor differences between ITLD and TDD1 tasks, i.e. ITLD task has 62.8%, 66.67% and 25.44% scores, whereas TDD1 task has 66.24%, 76.92% and 29.98% scores for mean, median and standard deviation respectively. We can see higher values with respect to mean and median of method coverage in ITLD task, whereas TDD1 task has a higher standard deviation. In other words, on average 81% (median: 90% and standard deviation: 23.14%) methods are covered in ITLD task, while 65.49% of methods (median: 73.21% and standard deviation: 29.56%) are covered in TDD1 task.

In comparison of ITLD with the second TDD task (TDD2), instruction coverage show differences in both tasks. Instruction coverage of ITLD is higher than TDD2. Coverage score of instruction in ITLD task in on average 71.8% (median: 83.18% and standard deviation: 26.99%) as compare to average score of TDD2 task 61.49% (median 68.26% and standard deviation: 35.84%). Branch coverage of ITLD task has higher scores in
mean and median but lower score in standard deviation. On average, 36.91% of branches (median: 32.50% and standard deviation: 34.47%) are covered in ITLD task, while 27.01% of branches (median: 00.0%, standard deviation: 41.50%) are covered in TDD2 task. In terms of line coverage, ITLD task has higher scores in terms of mean and median, while the standard deviation in TDD2 is higher. In other words, on average 75.92% (median: 81.75% and standard deviation: 23.30%) of lines are covered in ITLD task, on the other hand, on average 62.06% (median: 60% and standard deviation:33.66%) lines are covered in TDD2 task. However, minor differences found in cyclomatic complexity in terms of mean and median ITLD has higher scores, whereas, TDD2 task has higher score in standard deviation. In other words, on average 62.80% paths (median: 66.67% and standard deviation: 25.44%) are covered in ITLD task, on the other hand, TDD2 has on average 59.40% (median: 60.00% and standard deviation 32.07%) coverage scores. Same situation for method coverage scores where ITLD task has higher scores in terms of mean and median and TDD2 task has higher score in terms of standard deviation. In other words, on average 81.00% (median: 90% and standard deviation: 23.41%) of methods are covered in ITLD task, on the other hand, on average 61.41% (median: 60% and standard deviation: 31.01%) methods are covered in TDD2 task.

5.3 Descriptive statistics of mutation score

After removing, the projects with mutation score zero we came up with the descriptive statistics shown in Table 7.

Table 7. Descriptive statistics mutation

<table>
<thead>
<tr>
<th>Participants-count</th>
<th>Min.</th>
<th>Median</th>
<th>Mean</th>
<th>Max.</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITLD</td>
<td>2.00</td>
<td>62.00</td>
<td>54.69</td>
<td>83.00</td>
<td>27.94</td>
</tr>
<tr>
<td>TDD1</td>
<td>6.00</td>
<td>82.00</td>
<td>69.59</td>
<td>93.00</td>
<td>28.40</td>
</tr>
<tr>
<td>TDD2</td>
<td>2.00</td>
<td>35.00</td>
<td>27.93</td>
<td>59.00</td>
<td>22.32</td>
</tr>
</tbody>
</table>

To make a paired comparison between the mutation scores of different treatments, we chose participants whose mutation score could be measured in both ITLD and TDD1 tasks and presented the descriptive statistics for those in Table 8. Similarly, we selected participants whose mutation score could be measured in both ITLD and TDD2 tasks and provide the statistics for those in Table 9. Table 10 presents the statistics in terms of mutation score for TDD1 and TDD2 tasks.

Table 8. Descriptive statistics mutation score for ITLD – TDD1

<table>
<thead>
<tr>
<th>Participants-count</th>
<th>Min.</th>
<th>Median</th>
<th>Mean</th>
<th>Max.</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITLD</td>
<td>17</td>
<td>67</td>
<td>57.5</td>
<td>83</td>
<td>26.1672</td>
</tr>
<tr>
<td>TDD1</td>
<td>56</td>
<td>87.5</td>
<td>83.5</td>
<td>93</td>
<td>10.64842</td>
</tr>
</tbody>
</table>
We came up with 10 common participants’ data, for which we could measure mutation score for both ITLD and TDD1 tasks. We observe that, TDD1 scores better in terms of mutation score i.e. higher median and mean values, and lower standard deviation scores. This indicates that the unit test cases written in TDD approach could be able to detect more faults/defects than the unit test cases written in ITLD approach. To support our claim, we ran statistical tests on data and reported our findings in section 5.5 (hypothesis testing).

### Table 9. Descriptive statistics mutation score for ITLD – TDD2

<table>
<thead>
<tr>
<th></th>
<th>Participants-count</th>
<th>Min.</th>
<th>Median</th>
<th>Mean</th>
<th>Max.</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITLD</td>
<td>12</td>
<td>6</td>
<td>84</td>
<td>71.58</td>
<td>93</td>
<td>29.09689</td>
</tr>
<tr>
<td>TDD2</td>
<td>12</td>
<td>2</td>
<td>32.5</td>
<td>25.58</td>
<td>59</td>
<td>21.25798</td>
</tr>
</tbody>
</table>

We filtered 12 participants to compare mutation score for ITLD and TDD2 tasks. According to Table 9, this time ITLD task has higher values in terms of minimum, median, mean, max than TDD2 task. The difference between the two comparisons might be due to validity maturity test (described in section 7.2: limitation of study). Another reason might be complexity of second TDD task. Legacy code was provided to developers and it is sometime difficult for developer to understand and modify code than writing new code from scratch.

### Table 10. Descriptive statistics of mutation score for TDD1 - TDD2

<table>
<thead>
<tr>
<th></th>
<th>Participants-count</th>
<th>Min.</th>
<th>Median</th>
<th>Mean</th>
<th>Max.</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDD1</td>
<td>9</td>
<td>17</td>
<td>72</td>
<td>58.89</td>
<td>83</td>
<td>25.37934</td>
</tr>
<tr>
<td>TDD2</td>
<td>9</td>
<td>2</td>
<td>38</td>
<td>32</td>
<td>59</td>
<td>23.28089</td>
</tr>
</tbody>
</table>

Nine participants filtered to compare mutation score results in between TDD1 and TDD2 tasks. From Table 10, we observe that TDD1 task has higher mutation score than TDD2 task, which shows, unit test case written in TDD1 task has more defect detection abilities than unit test case written in TDD2 task.

### 5.4 Box plots

In this section, we present comparison of results for ITLD with first and second TDD tasks, in graphical view, with the help of box plots.

#### 5.4.1 Box plot analysis of code coverage

In this section, we describe box plot analysis of code coverage metrics.
Figure 4. Box plot of INScov

Figure 4 shows box plot of instruction covered, we can observe difference of median between ITLD and TDD1 tasks. TDD1 has the highest instruction coverage in terms of median. On the other hand, we observe higher instruction coverage in ITLD task compared to the TDD2 task. Furthermore, the range of INScov metric is much larger in TDD2 task, compared to the range in TDD1 and ITLD tasks (Figure 4).

Figure 5. Box plot of BRCov

Figure 5 shows a major difference between ITLD and TDD1 task in terms of median branch coverage (BRCov). TDD1 task has the highest branch coverage than ITLD and TDD2 tasks, whereas, the median of TDD2 is zero. Again, we see that TDD approach improves the BRCov values compared to ITLD in the first task, whereas the score dramatically decreases in the second task developed using TDD.
Figure 6. Box plot of LOCov

We observe small difference of line coverage (Figure 6) in terms of median between ITLD and TDD1 tasks, whereas, TDD2 task has lower median than ITLD and TDD1 task. Although, ITLD task has lower median score, it has a higher range than TDD1 task.

Figure 7. Box plot of COMCov

Figure 7 represents box plot for cyclomatic complexity coverage (COMCov) metric for ITLD, TDD1 and TDD2 tasks. We can observe difference of median in ITLD and TDD1 tasks. TDD1 task has higher median than ITLD and TDD2 tasks, while TDD2 task has higher range than both ITLD and TDD1 tasks (Figure 7).
5.4.2 Box plot analysis of mutation score

As described in Section 5.3, we performed paired comparison of three tasks in terms of mutation score metric. We follow the same logic and perform box plot analysis in pairs.
Figure 9. Box plot of MUTScore for ITLD and TDD1

We can observe difference in mutation score of ITLD and TDD1 tasks in box plot (Figure 9), in terms of median where TDD1 task has higher mutation score than ITLD task. This box plot also shows more range of mutation score for ITLD task.

Figure 10. Box plot of MUTScore of ITLD and TDD2

Figure 10 shows difference between ITLD and TDD2 tasks. ITLD task has higher medians of mutation score than TDD2 task.
5.5 Hypothesis testing

In this section, we present evaluation of our hypotheses via statistical testing using Mann-Whitney U test and Wilcoxon rank sum test. First, we perform evaluation on Class-Level metrics using Mann-Whitney U test followed by Wilcoxon rank sum test on Project Level metrics.

5.5.1 Comparison of ITLD and TDD

In this section, we provide the results of the tests at two different granularities consecutively for comparison of ITLD and TDD tasks.

**Class-Level Metrics**

Even though descriptive statistics and box plots show differences between ITLD and TDD1 tasks, we found that there are only two metrics (BRCov and MTCov) which we can conclude a significant difference between the two tasks (Table 11).

We could not reject the null hypothesis, i.e. there is no significant difference between ITLD and TDD1 tasks, for INSCov, LOCov and COMCov metrics.

**Table 11. Mann-Whitney U test results between ITLD and TDD1**

<table>
<thead>
<tr>
<th>Metric</th>
<th>p-value</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSCov</td>
<td>0.297</td>
<td>not rejected</td>
</tr>
<tr>
<td>BRCov</td>
<td>0.001</td>
<td>rejected</td>
</tr>
<tr>
<td>LOCov</td>
<td>0.989</td>
<td>not rejected</td>
</tr>
<tr>
<td>COMCov</td>
<td>0.330</td>
<td>not rejected</td>
</tr>
<tr>
<td>MTCov</td>
<td>0.008</td>
<td>rejected</td>
</tr>
</tbody>
</table>

In comparison of ITLD and TDD2 tasks (Table 12), we reject null hypothesis for method coverage ($p=0.006$), whereas, we could not reject null hypothesis for instruction, branch, cyclomatic complexity and lines of code coverage metrics (Table 12).

**Table 12. Mann-Whitney U test results between (ITLD) and (TDD2)**

<table>
<thead>
<tr>
<th>Metric</th>
<th>p-value</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSCov</td>
<td>0.418</td>
<td>not rejected</td>
</tr>
<tr>
<td>BRCov</td>
<td>0.060</td>
<td>not rejected</td>
</tr>
<tr>
<td>LOCov</td>
<td>0.112</td>
<td>not rejected</td>
</tr>
<tr>
<td>COMCov</td>
<td>0.634</td>
<td>not rejected</td>
</tr>
<tr>
<td>MTCov</td>
<td>0.006</td>
<td>rejected</td>
</tr>
</tbody>
</table>

**Project-Level Metrics**

In order to confirm results of Mann-Whitney U test on raw class level data, we applied Wilcoxon rank sum test at project level data for paired comparison. We aggregate raw class level data into project level data by calculating mean (MEAN), median
(MEDIAN), standard deviation (SD), minimum (MIN) and maximum (MAX) of each metric.

Mann-Whitney U test (Table 11) results between ITLD and TDD1 tasks shows that, for instruction coverage we could not reject null hypothesis, which is also same in Wilcoxon rank sum test results (Table 13). Null hypothesis is rejected for branch coverage in Mann-Whitney U test with \( p \) (0.001) (Table 11). Wilcoxon rank sum test results indicate we reject null hypothesis for mean, median and max branch coverage but for standard deviation and minimum branch coverage we have higher \( p \)-values (0.906) and (0.126) respectively (Table 13). Wilcoxon rank sum test result shows null hypothesis could not rejected for line coverage with \( p \) (mean=0.925, median=0.705, standard deviation=0.722, max=0.514 and min=0.421) (Table 13). For cyclomatic complexity, we could not reject null hypothesis as by Mann-Whitney U test results (Table 11) as well as in Wilcoxon rank sum test results (Table 13), for mean, standard deviation, maximum and minimum. However, there may be difference of results for method coverage as we reject null hypothesis in Mann-Whitney U test with \( p \) (0.008) (Table 11). We also observe significance difference of mutation score in between ITLD and TDD1 tasks since null hypothesis rejects with \( p \) (0.04) smaller than 0.05 (Table 13).

<table>
<thead>
<tr>
<th>Metric</th>
<th>( p )-value</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAN_INSCov</td>
<td>0.298</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_INSCov</td>
<td>0.118</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_INSCov</td>
<td>1.000</td>
<td>not rejected</td>
</tr>
<tr>
<td>MAX_INSCov</td>
<td>0.106</td>
<td>not rejected</td>
</tr>
<tr>
<td>MIN_INSCov</td>
<td>0.850</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEAN_BRCov</td>
<td>0.014</td>
<td>rejected</td>
</tr>
<tr>
<td>MEDIAN_BRCov</td>
<td>0.008</td>
<td>rejected</td>
</tr>
<tr>
<td>SD_BRCov</td>
<td>0.906</td>
<td>not rejected</td>
</tr>
<tr>
<td>MAX_BRCov</td>
<td>0.010</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_BRCov</td>
<td>0.126</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEAN_LOCov</td>
<td>0.925</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_LOCov</td>
<td>0.705</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_LOCov</td>
<td>0.722</td>
<td>not rejected</td>
</tr>
<tr>
<td>MAX_LOCov</td>
<td>0.514</td>
<td>not rejected</td>
</tr>
<tr>
<td>MIN_LOCov</td>
<td>0.421</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEAN_COMCov</td>
<td>0.218</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_COMCov</td>
<td>0.058</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_COMCov</td>
<td>0.722</td>
<td>not rejected</td>
</tr>
<tr>
<td>MAX_COMCov</td>
<td>0.205</td>
<td>not rejected</td>
</tr>
<tr>
<td>MIN_COMCov</td>
<td>0.962</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEAN_MTCov</td>
<td>0.065</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_MTCov</td>
<td>0.103</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_MTCov</td>
<td>0.813</td>
<td>not rejected</td>
</tr>
<tr>
<td>MAX_MTCov</td>
<td>0.249</td>
<td>not rejected</td>
</tr>
<tr>
<td>MIN_MTCov</td>
<td>0.074</td>
<td>not rejected</td>
</tr>
<tr>
<td>MUTScore</td>
<td>0.04</td>
<td>rejected</td>
</tr>
</tbody>
</table>
In comparison of ITLD and TDD2 tasks, our first hypothesis for instruction coverage could not be rejected in Mann-Whitney U test results (Table 12). If we confirm this result with Wilcoxon rank sum test results (Table 14), we can see null hypothesis could not be rejected for mean and median values but for standard deviation, min and max, null hypothesis is rejected. For branch coverage null hypothesis could not be rejected in mean ($p=0.298$) and standard deviation ($p=0.343$), but for median, mean and min values, null hypothesis is rejected. In comparison of ITLD and TDD2 tasks. Null hypothesis could not be rejected for line coverage, according to Mann-Whitney U test (Table 12). However, according to Wilcoxon rank sum test results (Table 14), null hypothesis could not be rejected for mean ($p=0.080$) and median ($p=0.130$) but rejected for standard deviation ($p=0.009$), max ($p=0.004$) and min ($p=0.000$). Null hypothesis of cyclomatic complexity could not be rejected for mean ($p=1.000$) and median ($p=0.887$). Whereas, hypothesis is rejected for standard deviation ($p=0.024$), max ($p=0.002$) and min ($p=0.000$) (Table 14). Null hypothesis for method coverage, is rejected for mean ($p=0.014$), median ($p=0.003$), standard deviation ($p=0.044$), min ($p=0.000$) and max ($p=0.014$); also rejected in Mann-Whitney U test. Null hypothesis of mutation score could be rejected with ($p=0.04$) (Table 14).

Table 14. Wilcoxon rank sum test score for ITLD – TDD2

<table>
<thead>
<tr>
<th>Metric</th>
<th>$p$-value</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAN_INS Cov</td>
<td>0.130</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_INS Cov</td>
<td>0.776</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_INS Cov</td>
<td>0.018</td>
<td>rejected</td>
</tr>
<tr>
<td>MAX_INS Cov</td>
<td>0.002</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_INS Cov</td>
<td>0.000</td>
<td>rejected</td>
</tr>
<tr>
<td>MEAN_BRCov</td>
<td>0.298</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_BRCov</td>
<td>0.001</td>
<td>rejected</td>
</tr>
<tr>
<td>SD_BRCov</td>
<td>0.343</td>
<td>not rejected</td>
</tr>
<tr>
<td>MAX_BRCov</td>
<td>0.000</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_BRCov</td>
<td>0.022</td>
<td>rejected</td>
</tr>
<tr>
<td>MEAN_LOCov</td>
<td>0.080</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_LOCov</td>
<td>0.130</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_LOCov</td>
<td>0.009</td>
<td>rejected</td>
</tr>
<tr>
<td>MAX_LOCov</td>
<td>0.004</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_LOCov</td>
<td>0.000</td>
<td>rejected</td>
</tr>
<tr>
<td>MEAN_COM Cov</td>
<td>1.000</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_COM Cov</td>
<td>0.887</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_COM Cov</td>
<td>0.024</td>
<td>rejected</td>
</tr>
<tr>
<td>MAX_COM Cov</td>
<td>0.002</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_COM Cov</td>
<td>0.000</td>
<td>rejected</td>
</tr>
<tr>
<td>MEAN_MTCov</td>
<td>0.014</td>
<td>rejected</td>
</tr>
<tr>
<td>MEDIAN_MTCov</td>
<td>0.003</td>
<td>rejected</td>
</tr>
<tr>
<td>SD_MTCov</td>
<td>0.044</td>
<td>rejected</td>
</tr>
<tr>
<td>MAX_MTCov</td>
<td>0.014</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_MTCov</td>
<td>0.000</td>
<td>rejected</td>
</tr>
<tr>
<td>MUTScore</td>
<td>0.04</td>
<td>rejected</td>
</tr>
</tbody>
</table>
6. Comparison between TDD tasks

Participants of experiment perform one ITLD and two TDD tasks (of different complexity levels). In previous section, we compare results of ITLD task with both TDD tasks. In this section, we perform additional analysis between TDD1 and TDD2 tasks to check possible effects of practice/learning and complexity of task on unit test case quality with respect to testing practice. This section starts with the comparison of descriptive statistics and box plot followed by hypothesis evaluation with Mann-Whitney U test and Wilcoxon rank sum test at significance level ($p < 0.05$).

6.1 Descriptive statistics:

In comparison of TDD1 task with the second TDD task (TDD2) (Table 6), instruction coverage shows differences in both tasks. Instruction coverage of TDD1 has higher scores in mean and median but lower score in standard deviation. The coverage score of instructions in TDD1 task is on average 76.06% (median: 90.77% and standard deviation: 32.03%) compare to average score of TDD2 task 61.49% (median 68.26% and standard deviation: 35.84%). Branch coverage of TDD1 task is higher than TDD2 task. On average, 66.48% of branches (median: 92.71% and standard deviation: 43.19%) are covered in TDD1 task, while 27.01% of branches (median: 00.0%, standard deviation: 41.50%) are covered in TDD2 task. In terms of line coverage, TDD1 task has higher scores in mean and median, while the standard deviation of TDD2 is higher. In other words, on average 72.84% (median: 85.44% and standard deviation: 31.30%) of lines are covered in TDD1 task, on the other hand, on average 62.06% (median: 60% and standard deviation: 33.66%) lines are covered in TDD2 task. However, minor differences found in cyclomatic complexity coverage in terms of mean and median, whereas, TDD2 task has higher score in standard deviation. In other words, on average 66.24% paths (median: 76.92% and standard deviation: 29.98%) are covered in TDD1 task, on the other hand, TDD2 has on average 59.40% (median: 60.00% and standard deviation: 32.07%) score in coverage analysis. Same situation for method coverage where TDD1 task has higher scores in terms of mean and median and TDD2 task has higher score in terms of standard deviation. In other words, on average 65.49% (median: 73.21% and standard deviation: 29.56%) of methods are covered in TDD1 task, on the other hand, on average 61.41% (median: 60% and standard deviation: 31.01%) methods are covered in TDD2 task.

After removing participant with zero mutation score and pairing them for descriptive statistics of mutation score for TDD1 and TDD2 tasks, we can see that the TDD1 task has higher mutation score than TDD2 task (Table 10).
Box plot (Figure 11) also shows significant difference of mutation score in both tasks. We can observe higher mutation score of TDD1 task in terms of median (Figure 11).

6.2 Hypothesis testing

In this section, we present statistical evaluation of hypotheses for our side analysis i.e. comparison of two TDD tasks.

6.3 Mann-Whitney U test

In Table 15, we present results of Mann-Whitney U test on TDD1 and TDD2 tasks. The results describe difference between branch coverage metric of TDD1 and TDD2 tasks only, in which null hypothesis is rejected ($p = 0.00$), for remaining metrics there is no significant difference found.

<table>
<thead>
<tr>
<th>Metric</th>
<th>p-value</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSCov</td>
<td>0.197</td>
<td>not rejected</td>
</tr>
<tr>
<td>BRCov</td>
<td>0.000</td>
<td>rejected</td>
</tr>
<tr>
<td>LOCov</td>
<td>0.316</td>
<td>not rejected</td>
</tr>
<tr>
<td>COMCov</td>
<td>0.307</td>
<td>not rejected</td>
</tr>
<tr>
<td>MTCov</td>
<td>0.393</td>
<td>not rejected</td>
</tr>
</tbody>
</table>

6.4 Wilcoxon rank sum test

Mann-Whitney U test could not reject null hypothesis for instruction coverage metric (Table 15). Wilcoxon rank sum test results indicate we reject null hypothesis for standard deviation, maximum and minimum, but for mean and median, we have higher $p$-values (0.052) and (0.072) respectively (Table 16). Null hypothesis for branch
coverage is rejected in Mann-Whitney U test as well as in Wilcoxon rank sum test. In comparison of TDD1 and TDD2 tasks, null hypothesis could not be rejected for line coverage, according to Mann-Whitney U test (Table 15). However, according to Wilcoxon rank sum test null hypothesis could not be rejected for mean ($p=0.072$) and median ($p=0.058$) but rejected for standard deviation ($p=0.004$), max ($p=0.002$) and min ($p=0.004$) (Table 16). Same situation for cyclomatic complexity coverage, according to Wilcoxon rank sum test null hypothesis could not be rejected for mean ($p=0.156$) and median ($p=0.108$) but rejected for standard deviation ($p=0.008$), max ($p=0.001$) and min ($p=0.004$) (Table 16). In case of Method coverage null hypothesis could not be rejected in Mann-Whitney U test (Table 15) and according to Wilcoxon rank sum test results null hypothesis could not be rejected for mean ($p=0.394$) and median ($p=0.083$) but rejected for standard deviation ($p=0.003$), maximum ($p=0.003$) and minimum ($p=0.004$). Mutation score null hypothesis is evaluated with Wilcoxon rank sum test. We could reject null hypothesis for mutation score with $p$ value 0.01 (Table 16).

Table 16. Wilcoxon rank sum test score of TDD1 - TDD2

<table>
<thead>
<tr>
<th>Metric</th>
<th>$p$-value</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAN_INSCov</td>
<td>0.052</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_INSCov</td>
<td>0.072</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_INSCov</td>
<td>0.003</td>
<td>rejected</td>
</tr>
<tr>
<td>MAX_INSCov</td>
<td>0.002</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_INSCov</td>
<td>0.004</td>
<td>rejected</td>
</tr>
<tr>
<td>MEAN_BRCov</td>
<td>0.001</td>
<td>rejected</td>
</tr>
<tr>
<td>MEDIAN_BRCov</td>
<td>0.001</td>
<td>rejected</td>
</tr>
<tr>
<td>SD_BRCov</td>
<td>0.007</td>
<td>rejected</td>
</tr>
<tr>
<td>MAX_BRCov</td>
<td>0.034</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_BRCov</td>
<td>0.009</td>
<td>rejected</td>
</tr>
<tr>
<td>MEAN_LOCov</td>
<td>0.072</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_LOCov</td>
<td>0.058</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_LOCov</td>
<td>0.004</td>
<td>rejected</td>
</tr>
<tr>
<td>MAX_LOCov</td>
<td>0.002</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_LOCov</td>
<td>0.004</td>
<td>rejected</td>
</tr>
<tr>
<td>MEAN_COMCov</td>
<td>0.156</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_COMCov</td>
<td>0.108</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_COMCov</td>
<td>0.008</td>
<td>rejected</td>
</tr>
<tr>
<td>MAX_COMCov</td>
<td>0.001</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_COMCov</td>
<td>0.004</td>
<td>rejected</td>
</tr>
<tr>
<td>MEAN_MTCov</td>
<td>0.394</td>
<td>not rejected</td>
</tr>
<tr>
<td>MEDIAN_MTCov</td>
<td>0.083</td>
<td>not rejected</td>
</tr>
<tr>
<td>SD_MTCov</td>
<td>0.003</td>
<td>rejected</td>
</tr>
<tr>
<td>MAX_MTCov</td>
<td>0.003</td>
<td>rejected</td>
</tr>
<tr>
<td>MIN_MTCov</td>
<td>0.004</td>
<td>rejected</td>
</tr>
<tr>
<td>MUTScore</td>
<td>0.01</td>
<td>rejected</td>
</tr>
</tbody>
</table>
7. Interpretation

In this section, we present interpretation of results and draw possible conclusions. This section also provides descriptions about limitations and threats to the validity of this research. In the end, we generalize results and compare these findings with existing studies.

7.1 Evaluation and implications of results

We performed an experiment to analyze the effects of TDD on unit test case quality in an industrial setting. The statistical results of the experiment shed light to the differences of some metrics between the tasks implemented using ITLD and TDD.

In accordance to our first research question, we evaluate the effects of TDD on code coverage and mutation score in comparison to ITLD. We compare five code coverage metrics between ITLD and TDD tasks. In terms of complexity and nature, both tasks share almost similar characteristics. Subjects implemented both tasks from scratch. The results indicate significant differences in mutation score, branch and method coverage between ITLD and TDD1 tasks. This result explains us that the development practice does have an effect on code coverage metrics. The results also indicate that TDD has a positive effect on branch coverage. The positive effects of TDD on branch coverage could be explained as inherent properties of TDD. In TDD, developers have to write test cases first and then implement the code that satisfy the conditions of test cases. This practice would result in an increase in the branch coverage of test cases. On the other hand, in ITLD, developers write and focus on production code first, and according to our observation, they may pay less attention to test case implementation. However, test cases written in ITLD cover more methods than test cases written in TDD. This could possibly be explained by developers’ attitude of testing practice. While writing tests in ITLD, developers create unit test cases that check the functionality of functions in the production code (usually implemented as methods in object-oriented programming). Thus, we observe an increase in the method coverage. On the other hand, in TDD developers pay more attention to writing test cases for user stories, rather than implementing functions directly. In terms of mutation score, our results indicate positive effects of TDD since TDD helped developers to improve the defect detection ability of unit test cases.

To shed more light on the effects of development practice on unit test case quality, we compare ITLD task with a second TDD task, which is more complex in nature, and subjects had to implement this task on an existing code base. The positive effect of TDD on branch coverage could not be observed in this case. The results indicate that test case written in ITLD cover more methods and are able to kill more mutants than test cases written in TDD. This variation on the results indicates that task complexity could affect the generalizability of the experiment results. Another possible explanation of this variation could be, in the views of validity threats, the experiment duration, i.e. subjects’ interest on TDD may have decreased in the second round of TDD implementation.
As per our side analysis, we compare the first TDD task with the second TDD task to see the impact of practice with test driven development. We have found statistical differences in mutation score and branch coverage. Interestingly, we observe that test cases written during first TDD task cover more branches and are able to kill more mutants than test cases written during second TDD task. These results support our claim that external factors i.e. task complexity and validity threats, could affect the performance of subjects and generalizability of the study.

7.2 Limitations of study

Validity of results is always a concern in experiments. We consider threats to validity of results based on the checklist presented in Wohlin et al., (2000), some of those threats are listed below.

Conclusion validity highlights those threats, which can direct to have wrong conclusions about relationship in between treatments and outcome results of experiments (Wohlin et al., 2000). Violated Assumptions of statistical tests are concerned with assumption for tests. To avoid such threats we used non-parametric tests (Mann-Whitney U test and Wilcoxon rank sum test). Fishing and the error rate explains the searching for specific results, researcher can be influenced result to have specific conclusion from the results (Wohlin et al., 2000), to avoid this threat, we executed two statistical tests (Mann-Whitney U test for raw data on class level and Wilcoxon rank sum test on derived data at project level). Reliability of measure is another threat to validity of results; results validity is dependent on reliability of measure (Wohlin et al., 2000). To avoid this threat we used two measure to test our phenomenon (code coverage and mutation score), and two different tools to measure these quality attributes. Reliability of treatment implementation, indicate risk of different implementation for different participant or occasion (Wohlin et al., 2000), we avoided this risk via as possible standard implementation.

Internal validity, concerns the results observed in the study and the true causes of those results (for example, specific result are caused by independent variables or some other factor is involved) (Madeyski, 2010). Results are high fragile to maturation validity threat. Experiment scheduled over five days (almost whole working day), this duration can cause negative or in positive (in terms of learning new things) behavior change towards experiment task. Testing, repeated test can experience this threat, as subject know about the test, they can behave differently each time (Wohlin et al., 2000). We avoided this threat by not showing results of test to participants. Validity threat of selection exists for this experiment as subject volunteered them self for experiments and according to Wohlin et al., (2000) compared with the whole population, volunteers are more suited for a new task as they are generally more motivated.

Construct validity is about relationship of theory or concept and results of the experiment, as described by Madeyski, (2010), to which extent measure reflect theory or concept correctly. Mono-Method Bias, experiments can be misleading if, we use single type of observation or measure which can result is measurement bias (Wohlin et al., 2000). We avoid this threat by using multiple measures for unit test quality. Hypothesis guessing, according to (Wohlin et al., 2000), if participants able to guess aim or hypothesis of experiments they might base their behavior (positively or
negatively) according to their attitude towards hypothesis. We minimized risk of this threat by hiding information about our hypothesis from subjects of the experiment. Fear of being evaluated can cause threat of “evaluation apprehension”, which can motivate participants to forge the results of experiment (Wohlin et al., 2000), this experiment subjects are not being evaluated on the bases of their performance in tasks.

External validity highlights the concerns which can limit capacity of generalization of study findings, from a sample population to larger population i.e. industrial practice (Madeyski, 2010; Wohlin et al., 2000). Interaction of selection and treatment: threat refer to population sample, if that sample doesn't represent the whole population for which we attended to generalize our findings (Wohlin et al., 2000). In our experiment, we avoided this threat by choosing subject from software industry for which we want to generalize our findings (details can be found in section 3.8).

7.3 Inferences

In order to compare code coverage of unit test cases, Causevic et al., (2012a), Causevic et al., (2012b) and Madeyski, (2012) compared only branch coverage in academic experiment. Their findings did not indicate significance difference of branch coverage. On the other hand, our findings indicate significant difference (Table 11) in branch coverage and method coverage of ITLD task and TDD1 task. TDD1 task have more branches covered than ITLD task. Whereas, ITLD task covered more methods than TDD1 task. In comparison of ITLD and TDD2 task, null hypothesis could be rejected only for method coverage (Table 12). Descriptive statistics describes, on average ITLD covered more methods than TDD2 task (Table 6). Box plot (Figure 8) also show higher median of ITLD task than TDD2 task.

This experiment shows difference of mutation score of unit test cases generated using ITLD and TDD coding practices (Table 13, Table 14). This result indicate that, in industrial setting test effectiveness could vary in different coding practices. Previous studies (Causevic et al., 2012a; Causevic et al., 2012b; Madeyski, 2010) concluded no significant difference of mutation scores of both practices in academic setting.
8. Conclusions and future work

In this study, we performed experiment in industrial setting to evaluate effects of coding practices on unit test case quality. We used two attributes i.e. code coverage and mutation score to measure unit test case quality, as used by Causevic et al., (2012a), Causevic et al., (2012b) and Madeyski, 2010.

The major difference is found in terms of the mutation score of tasks performed using incremental test last development approach (ITLD) and tasks performed by implementing test driven development approach (TDD). TDD 1 task has more mutation score than ITLD, which shows more defect detection ability of TDD. However, we found decrease in mutation score in second TDD task, which is more complex in nature compared to ITLD and TDD1 task. There could be multiple reasons for this decrement of mutation score including complexity and legacy code. According to threat to validity, duration of experiment can cause positive or negative attitude in participant towards tasks (maturation validity threat). As per code coverage analysis, we found difference in branch and method coverage in comparison of ITLD and TDD1 tasks. Difference found in method coverage metric in comparison of ITLD and TDD2 tasks. In comparison of TDD1 and TDD2 tasks, effect of practice is visible in branch coverage and mutation score only.

In this experiment, we learnt that other factors like task’s nature/complexity, experiment duration and participant interest towards task could affect the results. Although, we are able to find different result than previous studies conducted in academic environment, we cannot generalize these results for industry. Therefore, we need to conduct more studies with industry to reduce/eliminate effect of these factors.
9. Acknowledgements

This research is supported by FiDiPro (Finland’s Distinguished Professor Program), which is led by Prof. Natalia Juristo.
10. References


Kollanus, S. (2010, September). Test-driven development-still a promising approach?. In Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the (pp. 403-408). IEEE. doi:10.1109/QUATIC.2010.73


Appendices:

Appendix A1: Task description – BowlingScoreKeeper

The objective is to develop an application that can calculate the score of a single bowling game using TDD. There is no graphical user interface. You work only with objects and JUnit test cases in this assignment. You won’t need a main method. The application’s requirements are divided into a set of user stories, which serve as your to do list. You should be able to incrementally develop a complete solution without an upfront comprehension of all the game’s rules. Don’t read ahead, and handle the requirements one at a time in the order provided. Solve the problem using TDD, starting with the first story’s requirement. Remember to always lead with a test case, taking hints from the examples provided. Only when a story is done, move on to the next one. A story is done when you are confident your program correctly implements all the functionality stipulated by the story’s requirement. This implies all of your test cases for that story and all of the test cases for the previous stories pass. You may need to tweak your solution as you progress towards more advanced requirements.

1. **Frame** Each turn of a bowling game is called a **frame**. 10 pins are arranged in each frame. The goal of the player is to knock down as many pins as possible in each frame. The player has two chances, or **throws**, to do so. The value of a throw is given by the number of pins knocked down in that throw.

   **Requirement:** Define a frame as composed of two throws. The first and second throws should be distinguishable.

   **Example:** [2, 4] is a frame with two throws, in which two pins were knocked down in the first throw and four pins were knocked down in the second.

2. **Frame Score** An ordinary frame’s score is the sum of its throws.

   **Requirement:** Compute the score of an ordinary frame.

   **Examples:** The score of the frame [2, 6] is 8. The score of the frame [0, 9] is 9.

3. **Game** A single game consists of 10 frames.

   **Requirement:** Define a game, which consists of 10 frames.

   **Example:** The sequence of frames [1, 5] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] represents a game. You will reuse this game from now on to represent different scenarios, modifying only a few frames each time.

4. **Game Score** The score of a bowling game is the sum of the individual scores of its frames.

   **Requirement:** Compute the score of a game.

   **Example:** The score of the game [1, 5] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] is 81. **5. Strike** 102

A frame is called a **strike** if all 10 pins are knocked down in the first throw. In this case, there is no second throw. A strike frame can be written as [10, 0]. The
score of a strike equals 10 plus the sum of the next two throws of the subsequent frame.

**Requirement:** Recognize a strike frame. Compute the score of a strike. Compute the score of a game containing a strike. **Examples:** Suppose [10, 0] and [3, 6] are consecutive frames. Then the first frame is a strike and its score equals \(10 + 3 + 6 = 19\). The game [10, 0] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] has a score of 94.

5. **Spare** A frame is called a spare when all 10 pins are knocked down in two throws. The score of a spare frame is 10 plus the value of the first throw from the subsequent frame.

**Requirement:** Recognize a spare frame. Compute the score of a spare. Compute the score of a game containing a spare frame.

**Examples:** [1, 9], [4, 6], [7, 3] are all spares. If you have two frames [1, 9] and [3, 6] in a row, the spare frame’s score is \(10 + 3 = 13\). The game [1, 9] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] has a score of 88.

6. **Strike and Spare** A strike can be followed by a spare. The strike’s score is not affected when this happens.

**Requirement:** Compute the score of a strike when it’s followed by a spare. Compute the score of a game with a spare following a strike.

**Examples:** In the sequence [10, 0] [4, 6] [7, 2], a strike is followed by a spare. In this case, the score of the strike is \(10 + 4 + 6 = 20\), and the score of the spare is \(4 + 6 + 7 = 17\). The game [10, 0] [4, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] has a score of 103.

7. **Multiple Strikes** Two strikes in a row are possible. You must take care when this happens for the computation of the first strike’s score requires the values of throws from two subsequent frames.

**Requirement:** Compute the score of a strike that is followed by another strike. Compute the score of a game with two strikes in a row.

**Examples:** In the sequence [10, 0] [10, 0] [7, 2], the score of the first strike is \(10 + 10 + 7 = 27\). The score of the second strike is \(10 + 7 + 2 = 19\). The game [10, 0] [10, 0] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] has a score of 112.

8. **Multiple Spares** Two spares in a row are possible. The first spare’s score is not affected when this happens.

**Requirement:** Compute the score of a game with two spares in a row.

**Example:** The game [8, 2] [5, 5] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] has a score of 98.

9. **Spare as the Last Frame** When a game’s last frame is a spare, the player will be given a bonus throw. However, this bonus throw does not belong to a regular frame. It is only used to calculate the score of the last spare.
**Requirement:** Compute the score of a spare when it’s the last frame of a game. Compute the score of a game when its last frame is a spare.

**Example:** The last frame in the game [1, 5] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 8] is a spare. If the bonus throw is [7], the last frame has a score of 2 + 8 + 7 = 17. The game has a score of 90.

10. **Strike as the Last Frame** When a game’s last frame is a strike, the player will be given two bonus throws. However, these two bonus throws do not belong to a regular frame. They are only used to calculate score of the last strike frame.

**Requirement:** Compute the score of a spare when it’s the last frame of a game. Compute the score of a game when the last frame is a strike.

**Example:** The last frame in the game [1, 5] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [10, 0] is a strike. If the bonus throws are [7, 2], the last frame’s score is 10 + 7 + 2 = 19. The game’s score is 92.

11. **Bonus is a Strike** Further bonus throws are not granted when a game’s last frame is a spare and the bonus throw is a strike.

**Requirement:** Compute the score of a game in which the last frame is a spare and the bonus throw is a strike.

**Example:** In the game [1, 5] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 8], the last frame is a spare. If the bonus throw is [10], the game’s score is 93.

12. **Best Score** A perfect game consists of all strikes (a total of 12 of them including the bonus throws), and has a score of 300.

**Requirement:** Check that the score of a perfect game is 300.

**Example:** A perfect game looks like [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] with bonus throws [10, 10]. It’s score is 300.

13. **Real Game Requirement:** Check that the score of the game [6, 3] [7, 1] [8, 2] [7, 2] [10, 0] [6, 2] [7, 3] [10, 0] [8, 0] [7, 3] [10] is 135. Congratulations, you are done!
Appendix A2. Task description - MarsRoverAPI

Mars Rover API

- Develop an API that moves a rover around on a grid.
- You are given the initial starting point (x,y) of a rover and the direction (N,S,E,W) it is facing.
- The rover receives a character array of commands.
- Implement commands that move the rover forward/backward (f,b).
- Implement commands that turn the rover left/right (l,r).
- Implement wrapping from one edge of the grid to another. (planets are spheres after all)
- Implement obstacle detection before each move to a new square. If a given sequence of commands encounters an obstacle, the rover moves up to the last possible point and reports the obstacle.
- Example: The rover is on a 100x100 grid at location (0, 0) and facing NORTH. The rover is given the commands "ffrrff" and should end up at (2, 2)
Appendix A3. Task description – MusicPhone

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 Run As Java Application, then select App -> JUnit Test) and see three UI windows appear. The Player and GPS UIs are complete. The Recommender window is just a skeleton. You will implement the required functionality in the Recommencer class.

A2. Run SmokeTest (Select the project, right-click and Run As Java Application, then select SmokeTest.java) and see the green bar. Check the structure of the sample test in SmokeTest.java to get familiar with the application.

A3. 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 a public method with the signature public static double computeDistance (GeoPoint, GeoPoint, String) 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 kilometers (“km”) or miles (“mi”) as specified by the third parameter. 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 null.

The formula for the great-circle distance is:

\[
\text{radius} = \left(\text{degrees} \times \pi\right) / 180; \\
\text{a} = \sqrt{\left(\frac{\Delta \text{Lat}}{2}\right)^2 + \cos(\text{Lat}_1) \times \cos(\text{Lat}_2) \times \sin^2\left(\frac{\Delta \text{Lon}}{2}\right)}; \\
\text{Great Circle Distance} = 2 \times \text{arc}\sin(\min(0, a)) \times \text{Radius}; \text{where Radius is the Earth's radius in kilometers (6371.01)} \text{ or in miles (3958.76)}. \]

<table>
<thead>
<tr>
<th>GeoPoint1</th>
<th>GeoPoint2</th>
<th>units</th>
<th>Great-circle distance (in units)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0)</td>
<td>(6.283, 360)</td>
<td>km</td>
<td>0</td>
</tr>
<tr>
<td>(0, 0)</td>
<td>(0, 60)</td>
<td>km</td>
<td>667.70</td>
</tr>
<tr>
<td>(0, 0)</td>
<td>(360, 360)</td>
<td>mi</td>
<td>0</td>
</tr>
<tr>
<td>(0, 0)</td>
<td>(0, 60)</td>
<td>mi</td>
<td>4145.80</td>
</tr>
<tr>
<td>(0.630, -1.513)</td>
<td>(36.12, -66.67)</td>
<td>km</td>
<td>1793.55</td>
</tr>
</tbody>
</table>

Notes: Numbers in roman type are in degrees. Numbers in italics are in radians.
Appendix A4. MusicPhone project layout

**app** – Package containing the UI resources and classes that provide data from LastFM.

**commons** – Package containing the data models, interfaces and logic modules used by the MusicPhone UI and tests. You will implement the missing logic inside this project. You may not change any of the interfaces or the DeviceManager. You may extend the behaviour of the Recommender, but without changing its existing behaviour or its interface. You may add one or more classes here if required.

**commons.dataClasses** – Package containing the basic classes representing the entities present in MusicPhone.
commons.interfaces – Package containing the interfaces from that are implemented by the different parts of this system. You should not change any of those.

commons.xmlData – Package containing the dump of Last.FM response for the API necessary to solve the task in XML format. Those are useful for testing purposes.

commons.dataConnectors – Package containing the implementation of IConnector (LastFmXmlConnector) methods to parse the Last.FM XML files. You may want to instantiate such class for testing purposes.

gps, player and recommender – Packages containing the UIs for the application main components. The UI is bounded to a class (e.g. GpsUI.java and Gps.java) which is the implementation of the interfaces present in commons.interfaces.

What you should know before you start

Initialization of components

The application project creates concrete instances of IPlayer, IGps, and IRecommender objects. These instances persist when the application is running. application will set the Connector property of the Recommender object to an instance of LastFmXmlConnector, which implements the IConnector interface.

The IConnector interface

Defines access to XML data from Last.FM. The IConnector class you need to test your implementation with preloaded XML data is LastFmXmlConnector. This class has a 0-argument constructor To access the XML data from Commons, use the Connector property of the Recommender class.

The XML data for testing

Located in the XmlData folder of the Commons project. The LastFmXmlConnector class accesses the files in this folder.

DeviceManager.Instance

Provides singleton access to instances of the IPlayer and IGps objects. When these objects are instantiated by the Application, they register themselves with the DeviceManager.

Architecture
Review the block diagram on the next page.

You’ll spend most of your time implementing the bold classes

For testing your implementation, you may use the XML data

**UnitTests**

- You may instantiate and test any of the classes defined in **Commons** here.

- You may also instantiate the concrete classes `Player`, `Gps`, and...