UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Cornelia Efros

Lab Package: Mutation Testing

Bachelor's thesis (9 ECTS)

Supervisor: Dietmar Pfahl

Tartu 2017

Lab Package: Mutation Testing

Abstract: The aim of this thesis is to create new lab materials on the topic of mutation testing for the course "Software Testing (MTAT.03.159)" taught at the University of Tartu. The thesis gives an overview of the mutation testing technique, describes materials used in the 2017 spring semester labs, and reports on the students' evaluation of the lab package. Possible future improvements based on students' feedback are also presented.

Keywords:

Software testing, mutation testing, lab package

CERCS: P170, computer science, numerical analysis, systems, control

Praktikumimaterjal: Mutatsioontestimine

Lühikokkuvõte: Käesoleva bakalaureusetöö eesmärk on luua Tartu Ülikooli ainele „Tarkvara testimine (MTAT.03.159)" uued praktikumimaterjalid mutatsioontestimise teemal. Töö annab ülevaate mutatsioontestimisest ning kirjeldab loodud materjale, mis võeti kasutusele 2017. aasta kevadsemestril. Tudengite antud tagasiside analüüsi põhjal on valminud ka parandused edasiseks.

Võtmesõnad:

Tarkvara testimine, mutatsioontestimine, praktikumimaterjal

CERCS: P170, arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1. Introduction

Software testing is a process of evaluating a software system under development. It is one of the core elements of the software development lifecycle. There are many types of testing, several of which are introduced to students in the course “Software Testing (MTAT.03.159)” taught at the University of Tartu. This thesis focuses on only one of these methods, i.e., mutation testing.

“Software Testing” is a course mainly taught to the second-year bachelor's degree programme students. It consists of weekly lectures and a total of six labs which are in a correspondence with the lectures. Each lab briefly introduces a different testing method, often using an adequate tool. In the past, mutation testing was only covered in lectures. Since 2017 Spring semester, it has been added to the topics taught in labs.

Mutation testing is a way of testing where bugs (mutations) are seeded into the program and then the tests are run. If the tests fail, the mutations are killed. If not, then the mutations are alive. The quality of these tests can be assessed from the percentage of mutations killed. The aim of this methodology is to measure the quality of written tests as well as to guide the formation of new tests.

The purpose of this thesis is to compose new materials about mutation testing for the software testing course. Furthermore, in the scope of this thesis project, the new lab package was applied and the students’ feedback has been collected in order to suggest improvements.

The thesis is divided into five main chapters. An overall introduction into the software testing course as well as mutation testing is given in the second chapter. Chapter three provides a thorough explanation about the design of the lab, including tasks, schedule, and materials used. The lab execution is described in chapter four. The penultimate chapter provides the previously collected feedback and suggests possible further improvements. The final chapter provides a summary identifying the main outcomes about generated materials.

2. Background

The following sub-sections give an overview of the software testing course and the subject of the newly added lab on mutation testing.

2.1. The course “Software Testing (MTAT.03.159)” at the University of Tartu

The 3 ECTS course taught in the spring semester teaches various testing techniques in lectures and labs. As described in the course wiki page[1], the course addresses general ideas behind various types of testing. The course which took place in the spring semester of 2017/2018, consisted of 7 lectures and 6 labs. The topics for the labs in chronological order were as follows:

1. Issue Reporting
2. Black-Box & White-Box Testing
3. Mutation Testing
4. Document Inspection and Defect Prediction
5. Static Code Analysis
6. Automated GUI Testing (Regression Testing)

2.2. Mutation Testing

One of the possibilities for measuring the quality of software tests is mutation testing, a fault based testing technique [2]. This method is used both in evaluation of written test suites as well as in guidance to create new tests [4]. The main concept of mutation testing involves generating different faulty versions of a program using mutation operators and running tests against those variants of the code. The quality of a test suite is measured by the mutation score. A mutation score is a ratio of the number of detected bugs over a total number of seeded bugs [2]. To kill a mutant, the test that passes on the original program must fail. Mutants are alive if no tests fail.

Mutation testing proves to be a useful tool when testing small pieces of code. It is rather complicated to work with it in large software systems as a vast number of mutants are generated

[1] https://courses.cs.ut.ee/2017/SWT2017/spring

that are not simple to follow-up upon. However, it can lead the tester's attention to test cases that need improving as it is possible that test suites, which are not good at finding mutants are also poor at finding real faults [4].

One of the problems with mutation testing is that it is possible to create mutants that are behaviourally equivalent to the code under test. Such mutants are called equivalent and there are no tests that can be written to kill them. It will require a human to distinguish an equivalent mutant from a non-equivalent one.

There are many publicly available tools for mutation testing. MuClipse, Judy, Jumble, Jester, and the PIT tool to mention a few [5]. The PIT tool was chosen for the scope of this thesis for its ease of use and integration with both Eclipse[2] and IntelliJ[3] IDE. The PIT tool will be explained in more detail in Section 3.1.2.

Mutation Testing Terminology

- Killed mutant – mutant detected by tests
- Alive mutant – mutant not detected by tests
- Equivalent mutant – a mutant semantically equivalent to the code under test
- Mutation operator – operator that mutates the code in small ways

[2] https://eclipse.org/
[3] https://www.jetbrains.com/idea/

3. Lab Design

The following sub-sections describe the structure of the lab and the materials used to guide both the students and the lab assistants.

3.1. Lab Schedule

It is meant for the students to spend 8 student hours on each lab. The approximate schedule for the mutation testing lab is presented in the following list:

- Introduction to mutation testing and motivation (including the Code Defenders game) – 45 min
- Introduction to homework assignment (including tool installation, familiarization with PIT, and introduction to Lab Materials) – 45 min

3.2. Lab Activities

The students were conducted to main activities during the lab session, an online game (Code Defenders) in order to get acquainted with the basic ideas of mutation testing, and an introduction to the required homework assignment.

3.2.1. Activity 1: Code Defenders

The first task in the lab is to get familiar with the mutation testing concept. The Code Defenders[4] game is played for the purpose of introducing students the general idea behind creating and killing mutants.

According to Code Defenders home page, Code Defenders is a web-based game created "to make software testing education more enjoyable". It was developed at The University of Sheffield, in the United Kingdom. The purpose of the game is to teach students better testing proficiency as well as make testing fun in order to become better at software engineering. [3]

[4] http://code-defenders.org/

Basic Concepts of the Game

There are two types of players, attackers and defenders, who compete over a fragment of code (written in Java) under test [3]. The attackers' task is to insert small faults (mutants) into the code; defenders will write tests (e.g., JUnit) to detect the mutants. Although the game offers various game modes, the battleground mode is used to let all students participate in one game.

Rules

The following game rules are taken from the Code Defenders online page.
Written tests and inserted mutants may not:

- contain loops.
- make calls to System.*.
- contain new methods or conditionals.

Additionally, the tests can only contain two assertions each. Example of a test taken from the Code Defenders web page is shown on Figure 1, an example of a mutant is shown on Figure 2.

```
import org.junit.*;
import static org.junit.Assert.*;

public class TestElevator {
    @Test(timeout = 4000)
    public void test() throws Throwable {
        Elevator e = new Elevator(10, 2);
        e.addRiders(1);
        assertEquals(1, e.getNumRiders());
```

Figure 1. Example of a JUnit test.

Figure 2. Example of a mutant.

Playing the Game

In order to get started with the game, the participating students must first create a user account. To access the game, lab assistants must create a new “Battleground” as seen in Figure 3. Students will then be given a choice to choose a role, i.e. attacker or defender. For a successful game, half of the students should be attackers and the other half should be defenders. Attacking and defending happens asynchronously. This allows players to play without waiting for the other team to finish their move. Points are awarded for each successful mutant (those which are not killed) and test that kill mutants. The exact scoring system is described in the Code Defenders study [4].

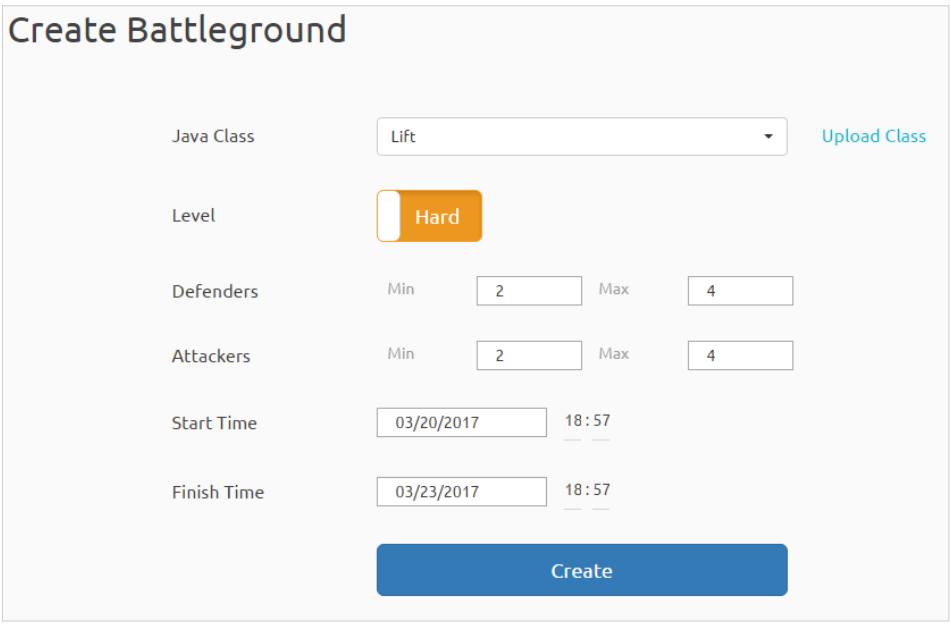


Figure 3. Creating a battleground.

It is possible to create equivalent mutants which cannot be killed by any tests. There is an option for the defenders to claim the mutant as equivalent if they believe it may be so (see Figure 4). In that case, the attacker who wrote the mutant should decide whether the mutant is equivalent or not. If it is, they should accept it. If not, they must prove it by writing a test that kills the mutant.

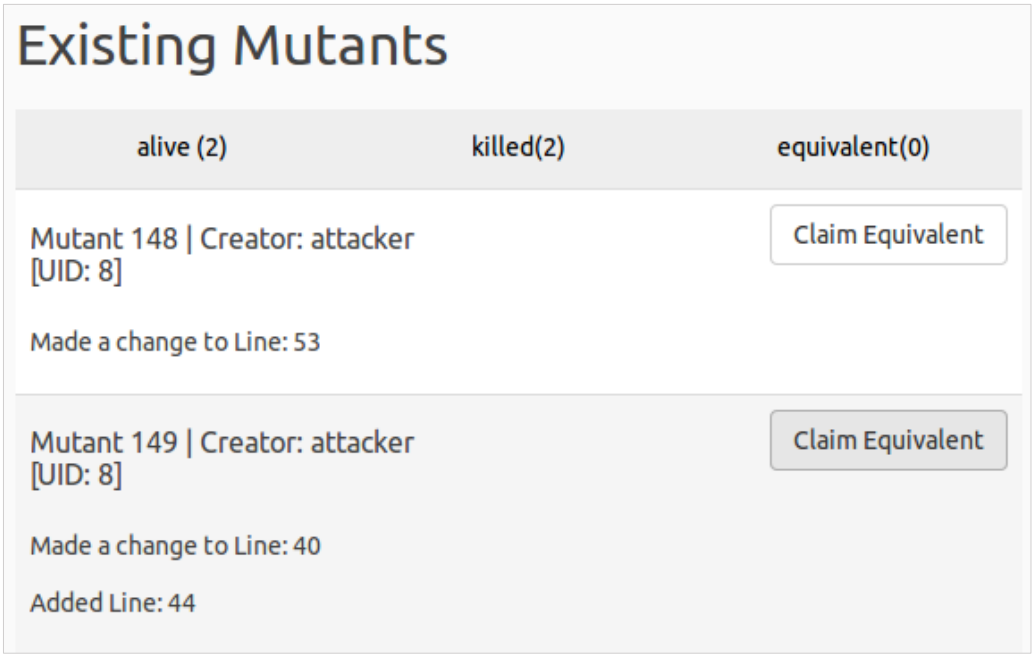


Figure 4. Claiming mutant as equivalent.

3.2.2. Activity 2: Homework Assignment

The second task is the homework where students are asked to work with the system under test. The aim of the homework is to introduce PIT tool, bring out the strengths and weaknesses of mutation testing and finally teach students to critically evaluate their tests. Students are also asked to give feedback on the lab to help make future improvements.

PIT Mutation Testing System

PIT is a tool created for measuring the quality of test suites, developed by Henry Coles, a software developer situated in Edinburgh[5]. PIT was chosen because it has a plugin for Eclipse IDE and IntelliJ IDE both of which are commonly used among students.

According to the PIT home page, the way PIT works is that it "applies a configurable set of mutation operators (or mutators) to the byte code generated by compiling your code". In other words, mutants are executed in byte code level after which mutants are put into the JVM and are therefore discovered by the test suite. To work with the PIT, Java 5 or above is required and either JUnit 4.6 or above or TestNG on the classpath, as said on PIT online page.

PIT is easy to use and additionally generates coverage reports, like line and mutation coverage. Moreover, it can be used with IntelliJ besides Eclipse as it resulted from previous years' labs that students would like IntelliJ to be supported [6]. On the downside, it would be difficult to use PIT when operating with large software systems as it generates thousands of mutants, which are hard to follow.

PIT includes various mutation operators; the default settings are used in lab. Mutation operators that are activated by default[6]:

- Conditionals Boundary Mutator
- Increments Mutator
- Invert Negatives Mutator
- Math Mutator
- Negate Conditionals Mutator

[5] http://pitest.org/about/
[6] http://pitest.org/quickstart/mutators/

- Return Values Mutator
- Void Method Calls Mutator (removes method calls to void methods)

All mutation operators except for Void Method Calls Mutator are listed in Appendix I.

System Under Test

For the purpose of the lab, a sample code has been developed. Precisely MinBinaryHeap.java, a minimum binary heap[7] implementation. With this program comes a test suite MinBinaryHeapTest.java with six test methods. After first running PIT, a report is generated which shows that the test suite has 71% mutation coverage and 94% line coverage.

Workflow

The overall goal is to kill mutants generated by PIT and find bugs in the code. The task given to students is to write more tests in addition to those six mentioned above, use PIT for generating mutants from the original code and thereby kill mutants and find bugs seeded into the original program. An example workflow is illustrated on Figure 5.

It is possible that PIT generates equivalent mutants which cannot be killed with tests. In that case, the students must point them out and shortly describe why they consider the mutants to be equivalent. Students can get up to 1 bonus point for finding equivalent mutants.

[7] https://en.wikipedia.org/wiki/Binary_heap

Figure 5. An example workflow.

Expected Outcome

It is necessary to submit a PDF-report containing:

- A list of found bugs and their fixes with explanations.
- A list of the added test cases.
- PIT mutation coverage and line coverage statistics.
- Feedback on the lab assignment.
- Bonus: A list of equivalent mutants with explanations (if found).

3.3. Lab Materials

Materials for the mutation testing lab are split into two sets: instructions for students, and for lab assistants. All the materials are found in Appendix II.

Students' instructions in the form of a PDF-document comprise three parts:

1. Short introduction to the mutation testing, the analysis tool, and the system under test.
2. Tasks:
   2.1. Playing Code Defenders
   2.2. Tool setup
   2.3. Homework about amending the test suite
3. Point distribution

The system under test and the instructions can be downloaded from the course wiki page as a ZIP-file.

Materials for the assistants additionally contain an accurate grading scheme and one version of the correctly finished homework.

3.4. Grading

Students will get a total of 10 points for each lab. For the mutation testing lab, the point distribution is as follows:

- 1 point for in-lab activity
- 1 point for finding bugs
- 1 point for additional test cases
- 5 points for an improved test suite
- 1 point for a fixed code
- 1 point for feedback

  Students can get up to 1 bonus point for pointing out equivalent mutants.

4. Lab Execution

The section below describes the implementation of mutation testing labs which took place in spring 2017.

The labs were held on 14th and 15th of March with a total of 5 lab groups consisting of up to 20 students. There was 1 assistant in each lab with a total of 3 assistants.

It was expected from the students to be already familiar with the topic before the beginning of the lab. The assistants briefly introduced mutation testing to avoid any inconveniences. After that, the Code Defenders game was shown and the students got some time to create a user and start playing. After about 30 minutes of playing the game, the students could start the homework and were able to ask guidance and questions from the instructor.

Two lab groups were examined to observe both time management and students' progress. Overall, most of the tasks went smoothly. However, there were two students who had problems with initially creating a user account at first. Additionally, dealing with equivalent mutants caused confusion among the students. On the positive side, two of the students discovered different ways to get the PIT plugin to work in IntelliJ, after which the corresponding instructions were added to the materials.

All in all, the majority of the students got 9 or more points from the lab as seen from the marks received. About half of the students also got points for the bonus task.

5. Feedback

This chapter presents the collected feedback and its analysis for improvements.

Every lab during the course asks for a constructive feedback on the materials, the execution of the lab, and positive and negative aspects found from the students. In the mutation testing lab, students were asked to fill in two different feedback forms: one as a last task in homework, and an online questionnaire. Students did not get marks for filling in the online questionnaire as it was voluntary. Moreover, the participation in the questionnaire did not affect their grades. Anonymous answers to the feedback given as a task in homework were only accessible to lab assistants and to the responsible lecturer. The questionnaire with summarized response count can be found in Appendix III.

89 students took part of the lab and 29 of them filled in the online questionnaire. The questionnaire was opened until the submission deadline of their assignment.

5.1. Collection of the Feedback

Qualitative Feedback

As the last part of the homework, the students were asked the following: "Try to be constructive and comment on the lab: was is (too) easy, (too) difficult, overall useful? How much time did you spend? How do you think it can be improved? Please be explicit and explain your answers."

Web-based Questionnaire

Students were also asked to fill in an online questionnaire using the software SurveyMonkey[8]. 29 of the students provided feedback. Evaluation was based on a Likert scale [7] which allows a person to express their attitude towards statements. There were 9 statements about mutation testing lab:

1. "The goals of the lab were clearly defined and communicated";
2. "The tasks of the lab were clearly defined and communicated";
3. "The materials of the lab were appropriate and useful";

[8] https://www.surveymonkey.com/

4. “The Code Defenders game was interesting and useful”;
5. “The PIT tool was interesting to learn.”
6. “If I have the choice, I will work with PIT tool again”;
7. “The support received from the lab instructors was appropriate”;
8. “The grading scheme was transparent and appropriate”;
9. “Overall the lab was useful in the context of this course”.

As the final question, the students were asked to write any additional feedback.

There were 5 options to choose from for statements 1-9: “Agree entirely”, “Somewhat agree”, “So-so”, “Somewhat disagree”, and “Disagree entirely”. An answer to statement 10 could be written in an open form.

5.2. Analysis

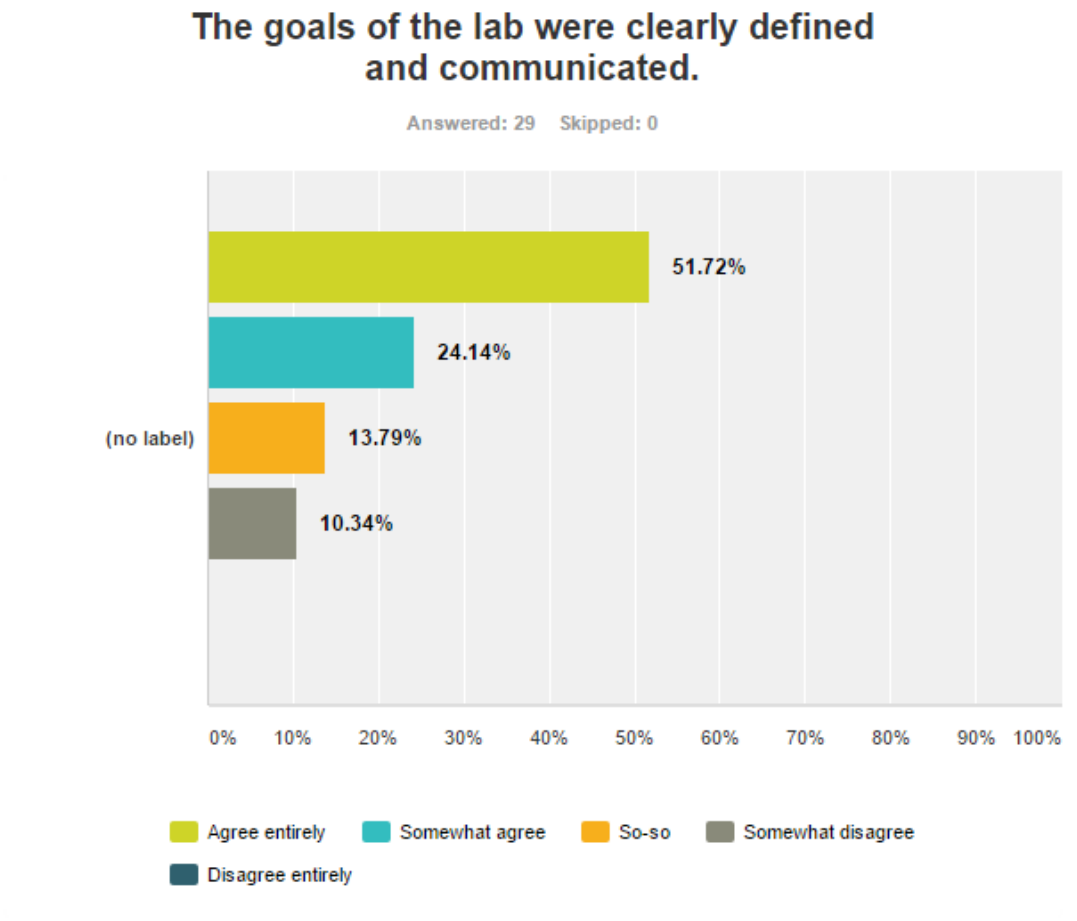
The feedback from students was mostly very positive, with a few students mentioning certain negative aspects.

Positive Aspects

Based on the qualitative feedback, many students said that the instructions provided were clear and easy to understand. As mentioned by a few of the students, pictures provided within the instructions were also helpful. It was said that playing Code Defenders was fun and the game made it clear how mutation testing works. According to some of the students, the PIT tool was also interesting, effective, and helpful. Participants mentioned that the difficulty level was somewhere near difficult but because it was interesting and required effort, it was not too hard. As can be seen from the following figures 6-14, qualitative feedback coincides with the online questionnaire.

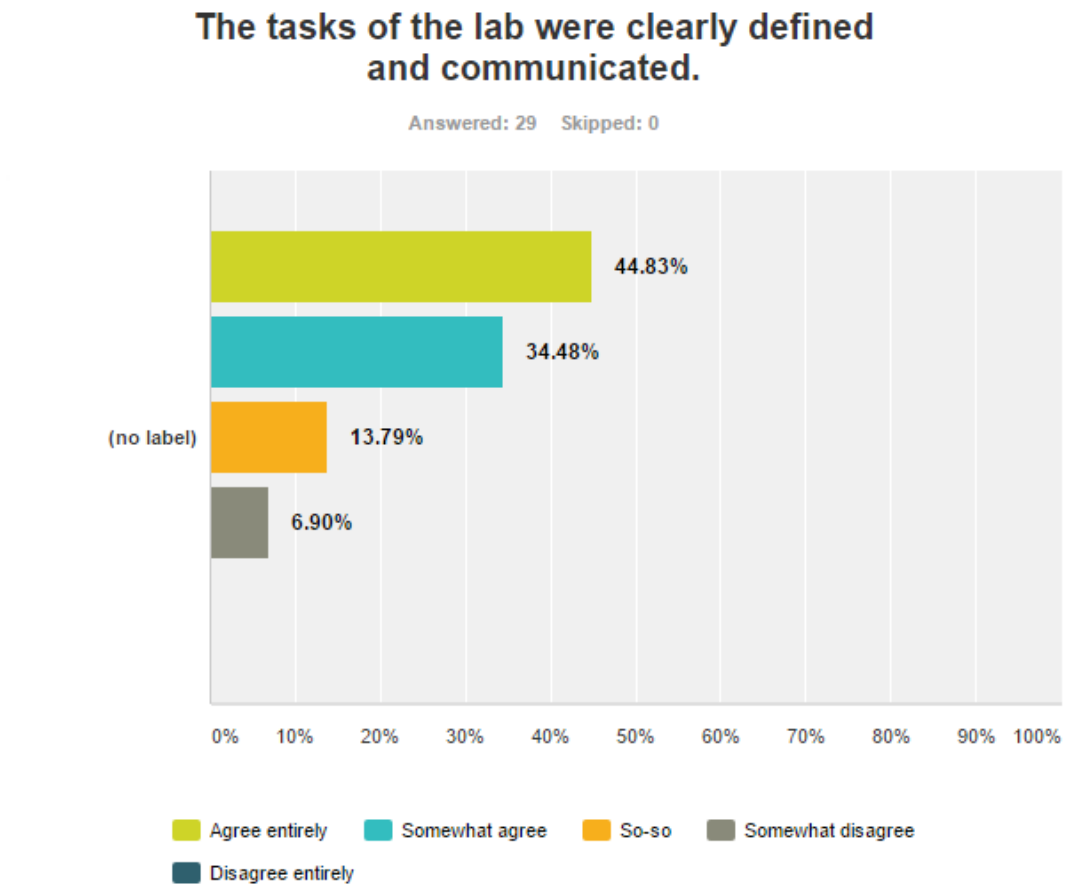
Three-fourths of the students agreed that the goals and tasks of the lab were clearly defined and communicated. About 70% of students found the Code Defenders game interesting and useful. Over 80% of students said it was interesting to learn about the PIT tool and the majority said they might use the tool again. All in all, over 80% of the respondents thought that the lab was useful in the context of the software testing course.

Students spent approximately 4 to 7 hours on the assignments. Faster students spent only 2 hours while others spent as much as 8 hours. Overall, the students' workload in terms of effort spent on the lab is comparable to that of the other labs in the course.



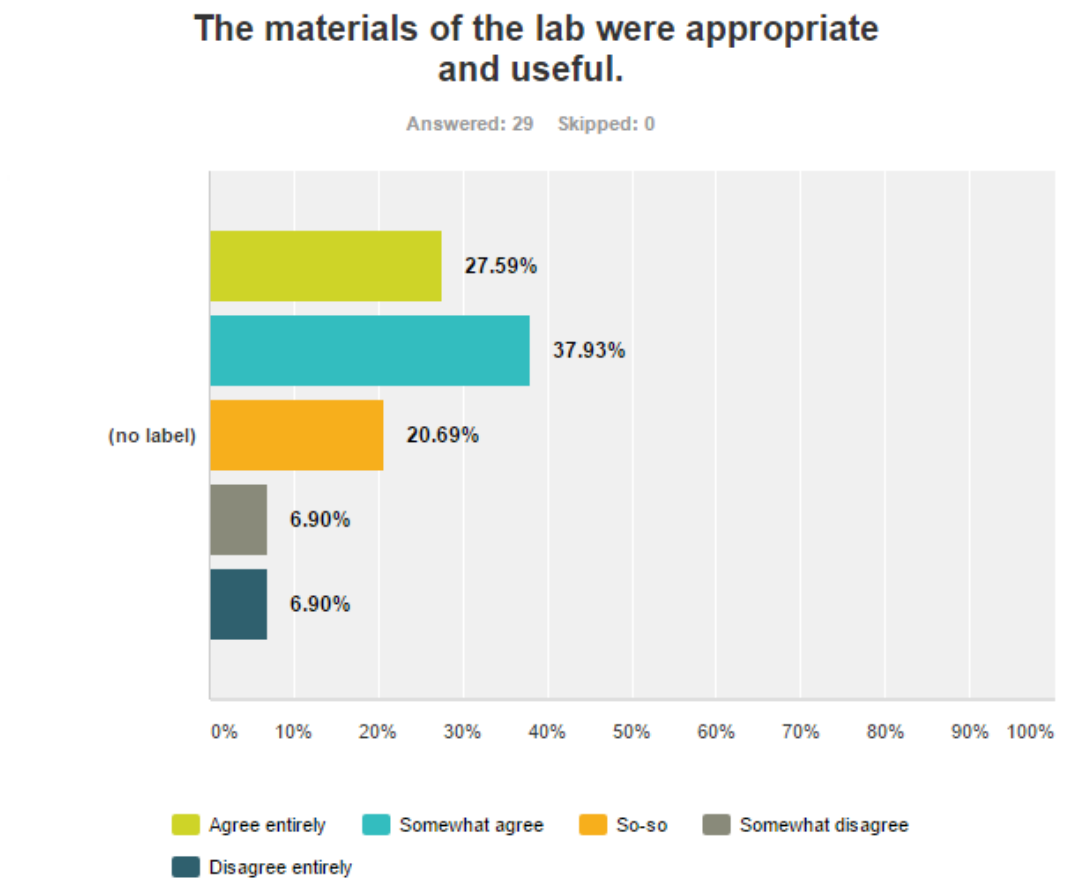
	Agree entirely (1)	Somewhat agree (2)	So-so (3)	Somewhat disagree (4)	Disagree entirely (5)	Total	Weighted Average
(no label)	51.72% 15	24.14% 7	13.79% 4	10.34% 3	0.00% 0	29	4.17

Figure 6. Results for the statement "The goals of the lab were clearly defined and communicated."



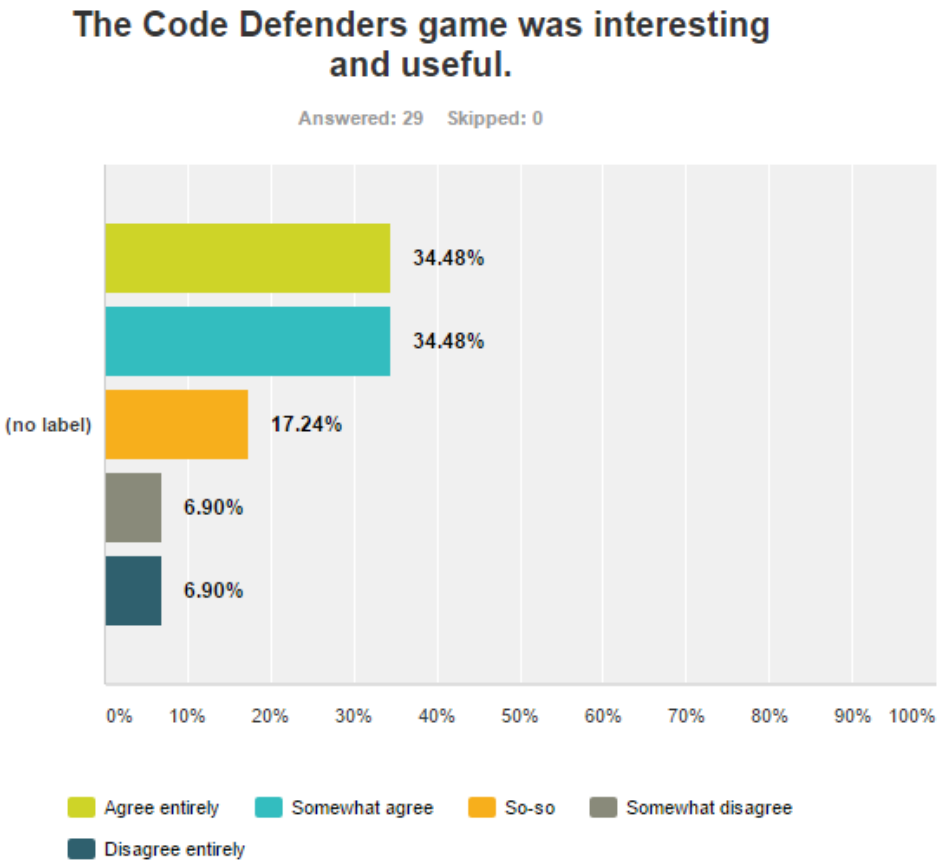
	Agree entirely (1)	Somewhat agree (2)	So-so (3)	Somewhat disagree (4)	Disagree entirely (5)	Total	Weighted Average
(no label)	44.83% 13	34.48% 10	13.79% 4	6.90% 2	0.00% 0	29	4.17

Figure 7. Results for the statement "The tasks of the lab were clearly defined and communicated."



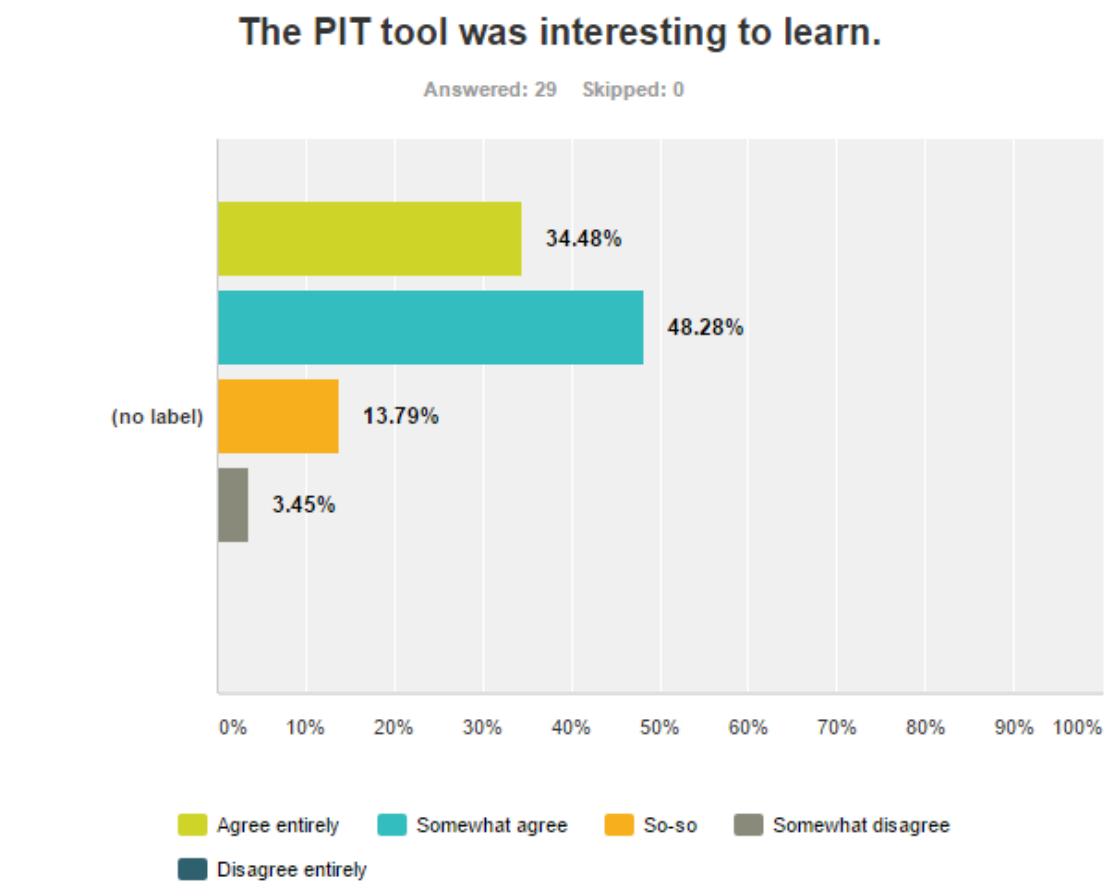
	Agree entirely (1)	Somewhat agree (2)	So-so (3)	Somewhat disagree (4)	Disagree entirely (5)	Total	Weighted Average
(no label)	27.59% 8	37.93% 11	20.69% 6	6.90% 2	6.90% 2	29	3.72

Figure 8. Results of the statement "The materials of the lab were appropriate and useful."



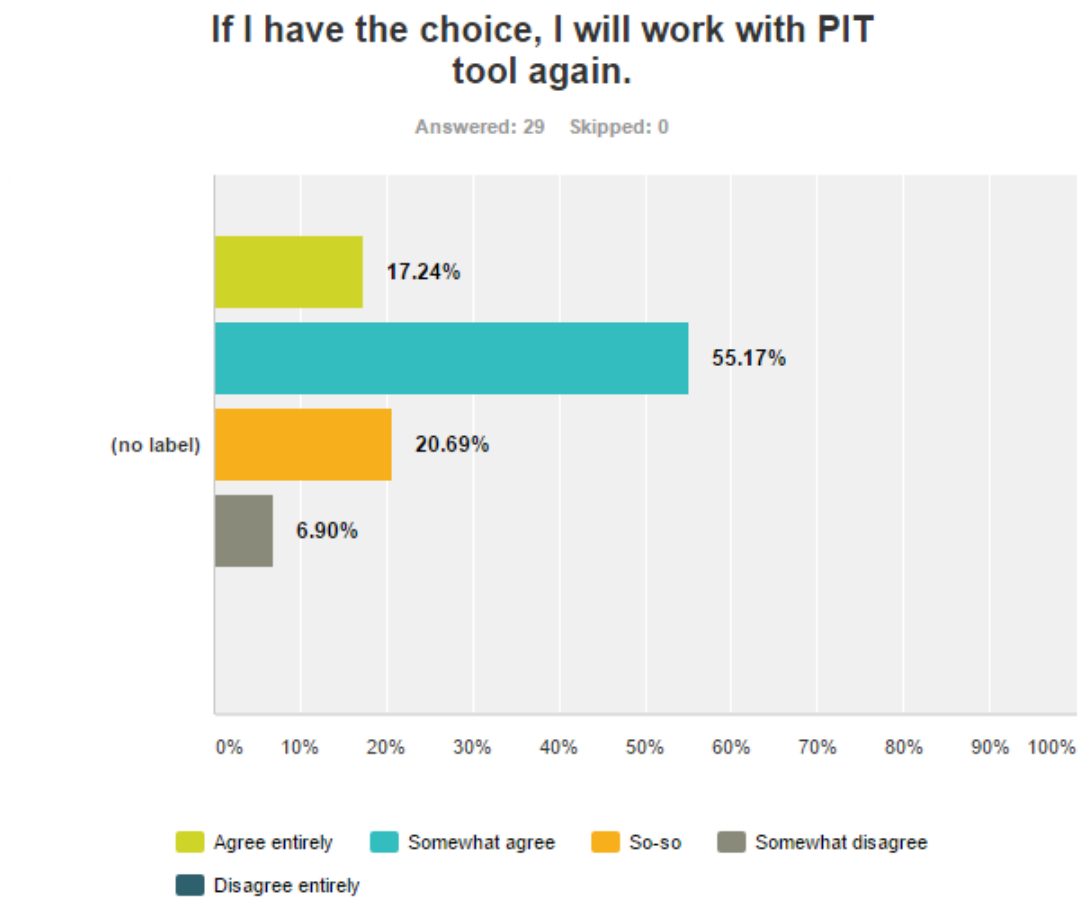
	Agree entirely (1)	Somewhat agree (2)	So-so (3)	Somewhat disagree (4)	Disagree entirely (5)	Total	Weighted Average
(no label)	34.48% 10	34.48% 10	17.24% 5	6.90% 2	6.90% 2	29	3.83

Figure 9. Results for the statement "The Code Defenders game was interesting and useful."



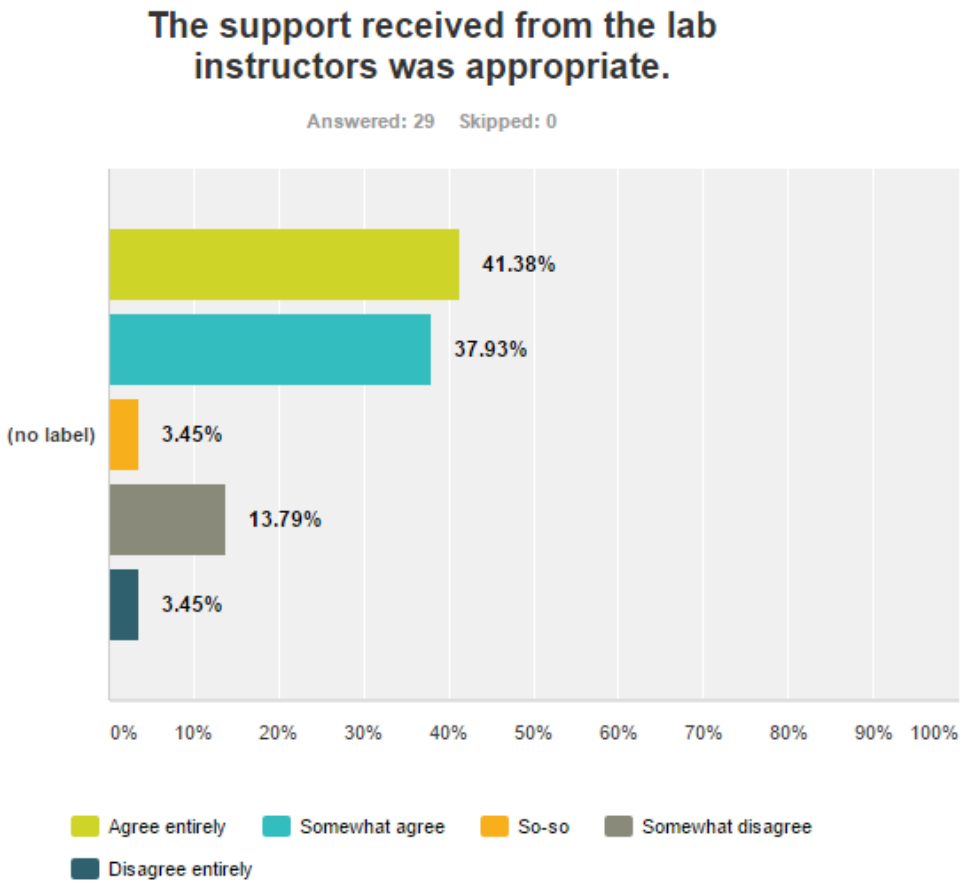
	Agree entirely (1)	Somewhat agree (2)	So-so (3)	Somewhat disagree (4)	Disagree entirely (5)	Total	Weighted Average
(no label)	34.48% 10	48.28% 14	13.79% 4	3.45% 1	0.00% 0	29	4.14

Figure 10. Results for the statement "The PIT tool was interesting to learn."



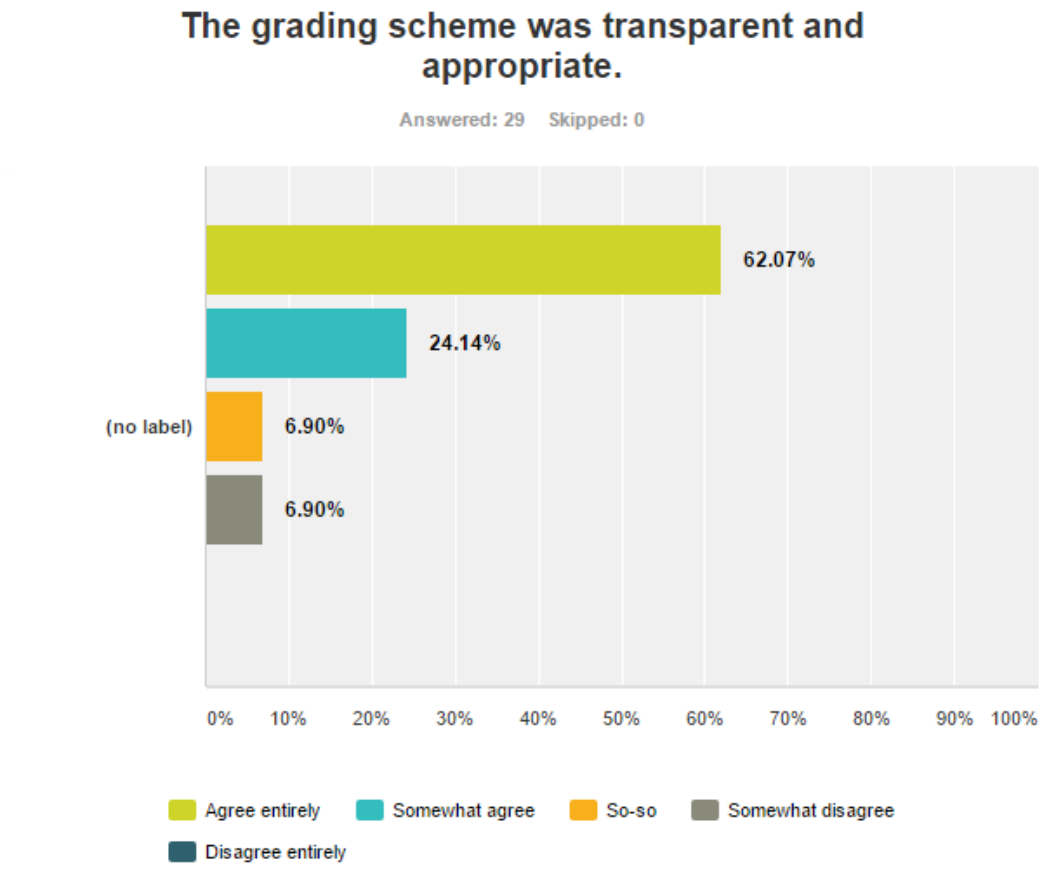
	Agree entirely (1)	Somewhat agree (2)	So-so (3)	Somewhat disagree (4)	Disagree entirely (5)	Total	Weighted Average
(no label)	17.24% 5	55.17% 16	20.69% 6	6.90% 2	0.00% 0	29	3.83

Figure 11. Results for the statement "If I have the choice, I will work with PIT tool again."



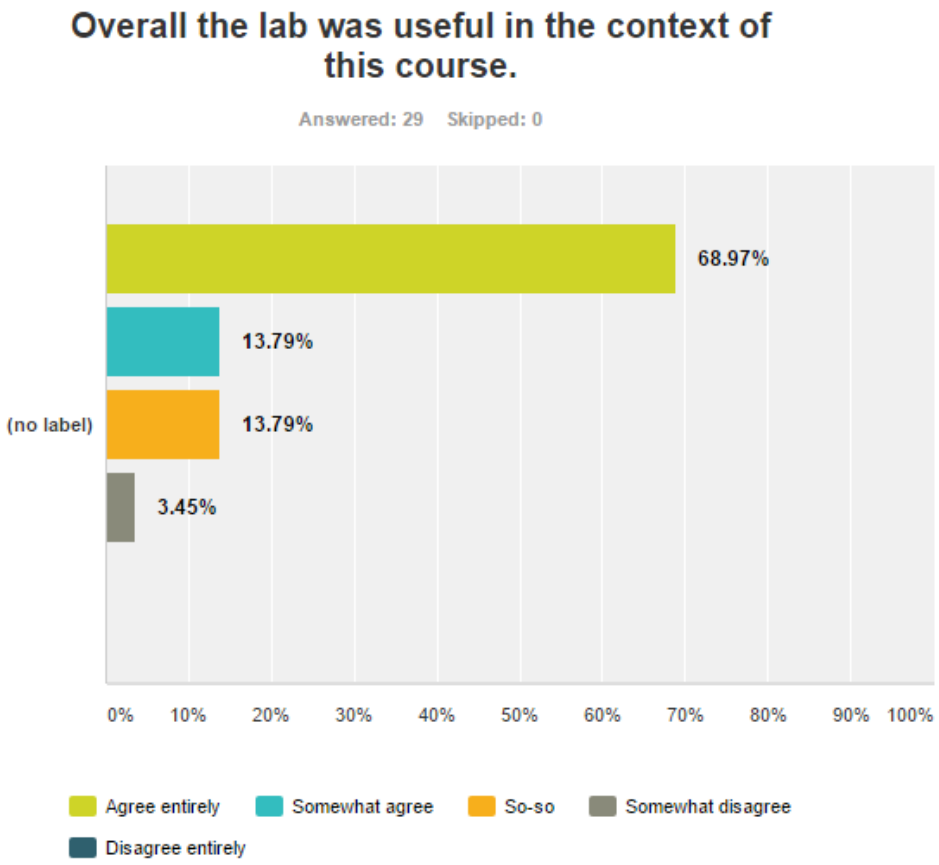
	Agree entirely (1)	Somewhat agree (2)	So-so (3)	Somewhat disagree (4)	Disagree entirely (5)	Total	Weighted Average
(no label)	41.38% 12	37.93% 11	3.45% 1	13.79% 4	3.45% 1	29	4.00

Figure 12. Results for the statement "The support received from the lab instructions was appropriate."



	Agree entirely (1)	Somewhat agree (2)	So-so (3)	Somewhat disagree (4)	Disagree entirely (5)	Total	Weighted Average
(no label)	62.07% 18	24.14% 7	6.90% 2	6.90% 2	0.00% 0	29	4.41

Figure 13. Results for the statement "The grading scheme was transparent and appropriate."



	Agree entirely (1)	Somewhat agree (2)	So-so (3)	Somewhat disagree (4)	Disagree entirely (5)	Total	Weighted Average
(no label)	68.97% 20	13.79% 4	13.79% 4	3.45% 1	0.00% 0	29	4.48

Figure 14. Results for the statement "Overall the lab was useful in the context of this course."

Negative aspects

Only a couple of students claimed that the instructions were hard to follow and the concept of mutation testing, especially the killing of mutants, was not sufficiently explained. A few of the students also mentioned that it was difficult to examine the code because of little or no knowledge about binary heaps. Some of the students, probably from the very first labs, suggested that there should be a guide for IntelliJ IDE as well but the guide was added after the first lab took place.

Some of the students also mentioned that it was not clear if they had to change tests that were already written or just write new tests due to unclear instructions.

5.3. Future Improvements

One of the possible improvements mentioned by the students, making a guide for IntelliJ IDE, was already made. An improvement can also be made within the lab instructions to make them more clear and understandable for all the students.
Many students pointed out that they should have the chance to try both attacking and defending when playing Code Defenders to try attacking as well as defending. This is an improvement to consider for the next year’s labs.

6. Conclusion

The main purpose of this thesis is to create new lab materials for the “Software Testing (MTAT.03.159)” course at the University of Tartu. The topic of the lab is mutation testing, a fault based software testing technique which evaluates software systems.

In the scope of the thesis, lab instructions for students and lab assistants were introduced. Materials consisted of two parts: lab activity (playing an online game) and homework (improve given test suite). Labs were implemented in 2017 Spring semester. Two types of feedback were asked from the students in order to analyse the materials, lab flow and make future improvements. The overall feedback was positive. Playing the game to learn mutation testing was fun, based on the students’ answers. They also pointed out that the homework part was relatively difficult, but interesting.

All in all, the materials given were satisfactory, although some improvements by students were suggested. For example, it was advised that the materials should be clearer to follow. Some recommendations regarding the web-based game were also mentioned.

References

[1] Course wiki page. [Online]. https://courses.cs.ut.ee/2017/SWT2017/spring (23.04.2017)

[2] M. Harman, Y. Jia, “An Analysis and Survey of the Development of Mutation Testing”, IEEE transactions on software engineering 37(5), 649–678 (2011)

[3] B. Clegg, G. Fraser, J. M. Rojas, “Teaching Software Testing Concepts Using a Mutation Testing Game,” In Proc. of the International Conference on Software Engineering Companion (ICSE), 2017, pp. 1-4.

[4] B. Clegg, G. Fraser, J. M. Rojas, T. White, “Code Defenders: Crowdsourcing Effective Tests and Subtle Mutants with a Mutation Testing Game,” In Proc. of the International Conference on Software Engineering (ICSE), 2017, pp. 1-12.

[5] S. K. Khatri, S. Rani, B. Suri, “Experimental Comparison of Automated Mutation Testing Tools for Java”, In Proc. of the 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), 2015, pp. 1-6.

[6] K. Leetberg (2016). “Lab Package: Static Code Analysis”, Bachelor's thesis, the University of Tartu.

[7] S. McLeod (2008), “Likert Scale”. [Online] https://www.simplypsychology.org/likert-scale.html (09.04.2017)

Appendix

I Mutation Operators

Table 1. Conditionals Boundary Mutator.

Original conditional	Mutated conditional
<	<=
<=	<
>	>=
>=	>

Table 2. Increments Mutator.

Original conditional	Mutated conditional
i++	i--
i--	i++

Table 3. Invert Negatives Mutator.

Original conditional	Mutated conditional
-i	i

Table 4. Math Mutator.

Original conditional	Mutated conditional
+	-
-	+
*	/
/	*
%	*
&	|
|	&
^	&
<<	>>
>>	<<
>>>	<<

Table 5. Negate Conditionals Mutator.

Original conditional	Mutated conditional
==	!=
!=	==
<=	>
>=	<
<	>=
>	<=

Table 6. Return Values Mutator.

Return Type	Original	Mutation
boolean	true	false
	false	true
int, byte, short	0	1
	other than 0	0
long	x	x-1
float, double	x	-(x+1.0)
	NAN	0
Object	non-null value	null
	null	throw java.lang.RuntimeException

II Lab Materials

Student Instructions

A1.1 “Lab 3 Instructions”, pdf-document
https://courses.cs.ut.ee/MTAT.03.159/2017_spring/uploads/Main/SWT2017-lab03.pdf

A1.2 “MinBinaryHeap”, zip-file
https://courses.cs.ut.ee/2017/SWT2017/spring/uploads/Main/lab3-code.zip

Lab Assistant Instructions

A1.3 “Lab 3 Instructions for TA”, pdf-document

A1.4 “MinBinaryHeapCorrect”, zip-file

For confidentiality reasons, lab assistant materials are not made available in the thesis but will be made available on request.

III Questionnaire Feedback

Statement	Agree Entirely (5)	Somewhat Agree (4)	So-So (3)	Somewhat Disagree (2)	Disagree entirely (1)	Weighted Average
The goals of the lab were clearly defined and communicated.	51.72%	24.14%	13.79%	10.34%	0.00%	4.17
The tasks of the lab were clearly defined and communicated.	44.83%	34.48%	13.79%	6.90%	0.00%	4.17
The materials of the lab were appropriate and useful.	27.59%	37.93%	20.69%	6.90%	6.90%	3.72
The Code Defenders game was interesting and useful.	34.48%	34.48%	17.24%	6.90%	6.90%	3.83
The PIT tool was interesting to learn.	34.48%	48.28%	13.79%	3.45%	0.00%	4.14
If I have the choice, I will work with PIT tool again.	17.24%	55.17%	20.69%	6.90%	0.00%	3.83
The support received from the lab instructors was appropriate.	41.38%	37.93%	3.45%	13.79%	3.45%	4.00
The grading scheme was transparent and appropriate.	62.07%	24.14%	6.90%	6.90%	0.00%	4.41
Overall the lab was useful in the context of this course.	68.97%	13.79%	13.79%	3.45%	0.00%	4.48

IV Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Forename Surname,

Cornelia Efros

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Title,

"Lab Package: Mutation Testing"

supervised by,

Dietmar Pfahl

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **11.05.2017**