UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Mari-Liis Pihlapuu

Software Testing: Integration Testing Lab Package

Bachelor's Thesis (9 ECTS)

Supervisor: Dietmar Pfahl, PhD

Tartu 2019

Software Testing: Integration Testing Lab Package

Abstract:
The aim of the given bachelor's thesis is to create materials for the Software Testing (LTAT.05.006) course in University of Tartu. The materials will be introduced to students in the spring semester of 2019. Feedback will be taken from students and other volunteers, which will be analysed, and areas of improvement will be marked.

Keywords:
Software testing, lab materials, integration testing, mocking

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

Lühikokkuvõte:
Antud bakalaureusetöö peaeesmärgiks on integratsioonitestimise materjalide loomine Tarkvara Testimise (LTAT.05.006) ainele Tartu Ülikoolis. Materjalid tutvustatakse tudengitele 2019. aasta kevadsemestril. Saadud tagasisidet analüüsitakse ja antakse parandusettepanekuid.

Võtmesõnad:
Tarkvara testimine, praktikumimaterjal, integratsiooni testimine, mocking

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Table of Contents

1 Introduction

Software testing is a part of the software development process to make sure that the software is working as intended. Today, there are many methods and tools released to help developers improve the quality assurance of their code. The University of Tartu introduces these possibilities to their students in their course Software Testing (LTAT.05.006).

The course Software Testing is a voluntary course in the bachelor's degree programme in the Informatics curriculum. As of the 2018/2019 spring semester, the course comprises seven lab packages and lectures, which introduce various ways of Software Testing along with materials and tools used to do so. The main goal of this thesis is to introduce the possibility of Integration testing to the second year Informatics students.

The motivation for this lab came from the possibility to introduce students integration testing to prepare for the possibility of needing these skills as a Software Engineer or a Quality Assurance Tester.

Integration testing is a testing method in which the interaction between separately developed units is tested to ensure that they are working as expected.
This thesis comprises five chapters. The first Chapter gives a brief introduction to integration testing and its possibilities. The second Chapter summarizes the materials and tools being used to execute the lab. The third Chapter describes details about the lab execution. The last chapter contains the feedback received from students and teacher's assistants along with analysis and suggestions for improvement.

2 Background Information

This section covers the overview of the Software Testing course and the main topics of the Integration Testing lab package.

2.1 Course Information

The Software Testing course (LTAT.05.006) is a voluntary 6 ECTS course in the bachelor's degree programme in the Informatics curriculum. It can also be attended by non-Informatics students, who have passed the compulsory subjects for this course.

As of the spring of 2019, the course consists of 11 topics [6], which are as follows:

1. Debugging
2. Basic Black-Box Testing
3. Combinatorial Testing
4. Basic White-Box Testing
5. Automated Web-Application Testing
6. Automated Integration Testing
7. Web-Application Testing in the CI/CD Pipeline
8. Automated GUI Testing
9. Mutation Testing
10. Static Code Analysis
11. Document Inspection and Defect Prediction

2.2 Integration Testing

Integration testing is a method of software testing in which the communication between individually developed components is tested. This method of testing is most useful when there are multiple microservices communicating with each other and to make sure that they produce expected results. In most cases, integration testing is done after unit testing and before validation testing [8].

2.2.1 Advantages

The advantages of this form of testing is the fact that the developer can validate whether the client's requirements are met with the newly developed software. Furthermore, while testing a Unit Under Testing (abbr. UUT), the components it will interact with do not require to be fully developed. Finally, it ensures that the modules of the software work in unity [8].

2.2.2 Disadvantages

Although it has many advantages, it also comes with a few disadvantages. Firstly, since Integration testing does not require other components which it interacts with, they need to be replaced with a fundamental amount of stubbing. Secondly, if the testing isn't planned well, then some critical faults in the system might get overlooked [8].

2.3 Mocking

According to Merriam-Webster dictionary, to mock is to imitate (someone or something) closely [7]. That is the precise action what different mocking tools in Integration Testing do, they mock the behaviour on many different objects or services that the UUT interacts with.

In unit testing, we test the class in isolation, but since the class is not particularly isolated – it is using services and methods from other classes – we need to simulate the real behaviour of these services and methods using different functionalities provided by mocking frameworks [8].

2.4 Integration Testing Tools

There are many options of tools available for developers to use in Integration testing. Small summary of some of the known tools, libraries and frameworks used today are the following.

2.4.1 JUnit

JUnit usually does not need introduction when it comes to Software Engineering. It is a simple, open source framework to write and run repeatable tests. Its features include: assertions for testing expected results, test fixtures for sharing common test data and test runners for running tests. Junit was originally written by Erich Gamma and Kent Beck [1].

2.4.2 TestNG

Alternative to JUnit is TestNG, which was inspired by JUnit, but it also introduced some new functionalities that were missing from JUnit. For example, you could test if your code was multithread safe and it provides flexible test configuration [2].

2.4.3 Mockito

Mockito is a mocking framework used for mocking objects, verification and stubbing. It is widely popular due to the fact they make tests very readable and produce clean verification errors. In late 2017 there was an analysis of top 1,000 GitHub projects and Mockito received a factual position of 4th most popular Java library across all libraries [3].

2.4.4 EasyMock

An alternative mocking framework is EasyMock. It is quite like Mockito – mostly because Mockito started off as an EasyMock fork [5] –, great resources for available for developers, but lacks the simplicity Mockito provides.

2.4.5 MockWebServer

MockWebServer is what its name suggests – a mocked web server. This library is relevant in this lab because the service under testing directly communicates with another server that is out of our control.

2.4.6 Tools Included in Lab Package

Out of these libraries, this thesis will focus on JUnit and Mockito since they are one of the most widely used frameworks available in the current time and are very simple to use by a beginner.

3 Lab Design

This section gives a brief overview of the materials created for the "Integration Testing" lab.

3.1 Lab Schedule

- The introduction to integration testing and mocking and tools to be used in the session should take approximately 60 minutes.
- The two provided lab tasks should take approximately 15-30 minutes, depending on the experience of the student.
- Introduction of the homework assignment and further questions takes approximately 20 minutes.

3.2 Lab Materials

The following materials are provided by the author of the thesis for the lab "Integration Testing".

- Documentation containing everything required for this lab session – from introduction to integration testing and the service under testing to the example use cases of the testing frameworks that are to be used in the lab session.
- Operator Communicator Java Project – Lab Version
    - Lab version for students, which contain only two test cases[1].
    - Lab version for lab supervisors, which contain full test coverage.
- Operator Communicator Java Project – Homework Version
    - Homework version for the students with no test coverage.
    - Homework version for the lab supervisor with all possible test cases.

The materials could be found in Appendix I. The documentation contains information about Integration Testing, service that will be under testing, lab and homework assignment, and how the students' homework will be graded.

3.3 Operator Communicator

Operator Communicator is a mobile payment service. A user has given their consent to a merchant (e.g. a gaming company) to charge the user, so they can receive extra content from their product. This type of service was chosen because it's similar to routine work the author of this thesis in mobile payments company Fortumo as a Software Engineer and it provides a simple overview on how to use mocking frameworks in case the service under testing is communicating with other services. The project was developed using Gradle and Java programming language.

Students will receive two different versions of the project – lab version and homework version. The homework version contains more complexity and is significantly different to the lab version of the project. In the lab, the students must write two additional test cases based on the information given

[1] Students have to write two more test cases as part of a practical task in the lab.

by the task set up and write 13 tests for the homework version based on the knowledge they have gained from the lab.

3.4 Lab Session Tasks

The purpose of the lab session tasks are as follows:

- Give the students an introduction to Integration Testing and mocking
- Introduce the students to possibilities Mockito and MockWebServer provide
- Brief introduction to the homework assignment

The lab should begin with the lab supervisor giving the students a simple introduction to Integration testing and explanation of the tools we will be using (e.g. mocking). Preferably showing the students some use case examples.

Next, the lab supervisor introduces the students the service to be tested, which is Operator Communicator. The lab supervisor will explain the full flow of the service: what it communicates with, what services are developed, and all the business requirements set. This is highly recommended, although a very detailed description is provided under the Figure 1 in the documentation (see Appendix I).

After the flow is explained, the students will open their projects that they downloaded from the Software Testing Lab course page along with the document provided with this lab. It is highly recommended to do this before the lab, so no issues would arise in the lab. The lab supervisor will explain the technical flow and how to decide what to mock.

Afterwards the students will be introduced two new tools and their most common usability: MockWebServer and Mockito. These tools were chosen as they are some of the most useful tools to use in Software Testing. There is a possibility to do the full test coverage with MockWebServer, but as it would be much more useful if the students learned more than one tool, Mockito was also chosen. It will be explained in what cases they are used and what are some of the possibilities it provides. Examples are also provided. Lab supervisor will also go through examples that are provided in code of the project they have installed and try to understand why the way of mocking was chosen for those cases. If there is time, students will do two exercises for “failure cases” as the project only contains tests for success cases.

By now students should have the conceptual and practical idea of what is integration testing and how and when to mock components and different elements. The lab supervisor will explain the students what they would have to do for home assignment.

3.5 Homework Task

Students are provided with the technical flow of the homework assignment, project without tests and “up-to-date” API of the operator. Based on the information they have learned in the lab session, they need to write at least 13 test cases for the homework version of the project to get full marks.

3.6 Grading

Students may receive maximum 10 points from the lab. The grading criteria is as follows:

- 0.65 points for each test that uses the tool we expect them to use, 0.35 will be provided if another tool is used, but the test is correct.
- Another 1.55 points will be provided if all the correct assertions and verifications using ArgumentCaptor are done, 0.5 points if some assertions and verifications are done, 0 points if no assertions or verifications are done.

If more than 13 test cases are provided, it is still treated as 13.

4 Lab Execution

The Integration Testing labs were executed on the 26th and 27th of March 2019. Since the homework assignment was unfinished at the time of the labs, only the lab task was included. It was decided that the students will get 2 bonus points for attending the lab. The reason for this was that the lab was planned too early in the course and the homework assignment needed more time to be polished. Overall 61 students attended the labs.

The first part of the lab was executed well. The lab supervisors were very well prepared and gave students all the necessary information needed before proceeding with the practical part of the lab. One of the lab supervisors had very detailed figure showing how Integration Testing works. As the lab package only focuses solely on the practical part of the lab, the idea of explaining the differences between unit testing and integration testing seemed something the lab package should improve upon.

The second part of the lab was executed differently, depending on the lab supervisors. One lab supervisor took on the strategy of first explaining, then letting the students try to work out their tasks by themselves before showing how it is done while the other lab supervisor had a more hands on approach as the experience level of the students was varying. Some students understood straight away what they were supposed to do and left the lab as soon as their practical part was done, but most inexperienced students needed more hands-on approach.

However, the practical part did not go as smoothly as had hoped. There was some confusion regarding what the task sees as a “failure case”. In the task it was expected that the students had to take on a case where an exception was thrown in the class under testing, but some students seemed to think that some values needed to be changed instead. This is also something that should be improved upon to provide students as much clarity as possible to avoid confusions like this in the future. Also, one student could not participate at all since their MacOS operating system could not work well with Gradle – there should be a forewarning for the students that they might have issues when they use different operating systems.

The timing of the lab went just a bit over 1 hour in average, depending on the number of students. Smaller class took less than 1 hour while a class with around 30 students took the full time of the lab. This timing was expected as the lab was executed without the homework assignment which would have needed a thorough introduction, so the students could do their homework efficiently.

Overall, the labs went very smoothly. Some students seemed to find the lab interesting but provided some feedback that the lab should include some smaller tasks to understand the tools better before taking on the practical assignment.

5 Feedback

This section of the thesis includes an overview of student-given feedback, which was analysed and suggestions for future improvements were made by the author.

5.1 Feedback Analysis

The feedback was taken from 61 students. They were asked 7 multiple-choice questions, answers ranging from "strongly disagree" to "strongly agree". Results were as follows (see Figure 1).

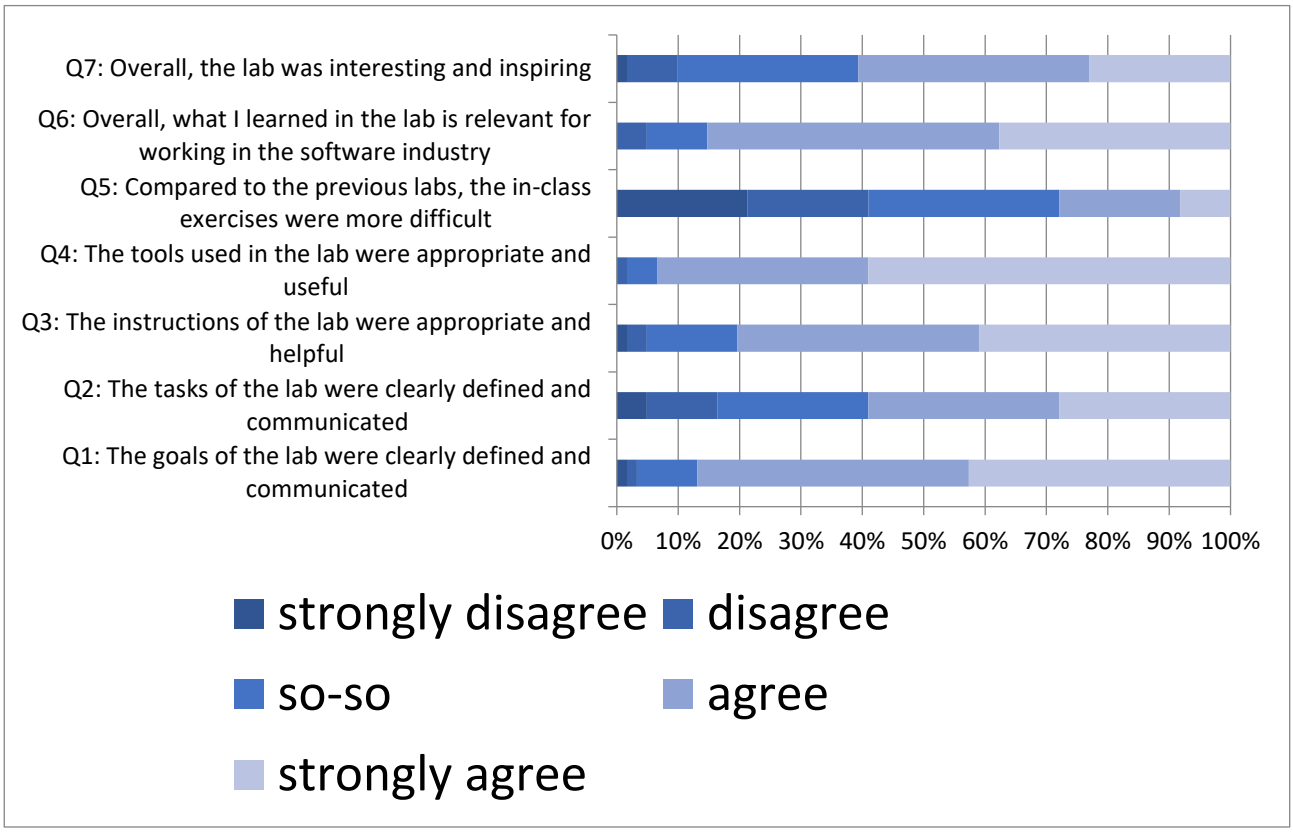

Figure 1. Questionnaire questions and results

The first question asked students whether the goals of the lab were clearly communicated and defined. 87% of the students agreed or strongly agreed with the statement while 13% of the students were not quite sure or disagreed.

Question two asked students whether the tasks of the lab were clearly defined and communicated. 59% of the students agreed or strongly agreed with the statement, 25% were not sure of the answer and 16% of the students disagreed or strongly disagreed. Although the percentage of satisfied students was higher, it was lower than initially expected, which indicates that the lab tasks should be reviewed and improved if possible.

Question three asked students whether the instructions of the lab were appropriate and helpful. 80% of the students agreed or strongly agreed with the statement, 15% were not sure of the answer while

5% disagreed or strongly disagreed, which might indicate that the instructions provided for the students were of satisfactory.

Question four asked the students whether the tools they were provided with were appropriate and useful. 93% of the students agreed and strongly agreed with the statement, 5% were not sure of the answer and 2% disagreed or strongly disagreed. From this feedback a conclusion could be made that the tools taken in are appropriate and do not have to be changed in the future.

Question five asked the students to provide feedback if the in-class task they were given was more difficult than in the previous classes. 28% of the students agreed or strongly agreed with the statement, 31% of the students were not sure and 41% of the students disagreed or strongly disagreed. This could mean multiple things: the lab was too easy for most of the students or the information they were given was enough that they understood their task in this lab better than in the previous labs.

Question six asked the students whether they thought that what they learned in the lab was relevant for working in the software industry. 85% of the students agreed or strongly agreed, 10% of the students were not sure and 5% of the students disagreed. This could mean that the students were either aware of the value of the given tools in the software industry or the lab gave enough information to the students that they concluded themselves at the end of the lab.

Question seven asked the students whether the lab was interesting or inspiring. 61% of the students agreed or strongly agreed with the statement, 30% of the students were not sure and 9% of the students disagreed or strongly disagreed. Although most students thought the lab was inspiring, the result is not as satisfactory. Perhaps including more interesting approach to learning the tools would improve the interest in the future.

In conclusion, most students were satisfied with the lab. The lab was not difficult, and they felt like they learned something relevant for working in the industry. Students were satisfied with the goals and the instructions given to them, but the tasks should be improved or made a bit more interesting.

5.2 Future Improvements

Based on the students' and lab supervisors' feedback, following improvement suggestions were collected:

1. Give the students more possibility to play around with the different functionalities Mockito and MockWebServer provides
2. Explain the difference between unit testing and integration testing
3. Make the project set up prerequisite for this lab to avoid any time going to waste
4. Clarify the tasks given to avoid confusion among the students
5. Simplify class and method naming in the project

The author of this thesis will improve the lab package based on points 3 and 4 as it would make the tasks clearer and would not require the presence of the author to explain to the students what is required to be done. Other improvement points may or may not be considered.

6 Summary

The aim of the given bachelor's thesis was to create materials for the Software Testing (LTAT.05.006) course in University of Tartu. The materials were introduced to students in the spring semester of 2019. Feedback was taken from students and other volunteers, which was analysed, and areas of improvement were marked.

The overall feedback for this lab was positive, but some areas of improvement were brought out. These notices should be taken in and improved upon in the future.

Based on the feedback, the lab package can be deemed useful and appropriate to include in the "Software Testing" course in the future.

7 References

[1] "JUnit FAQ," [Online]. Available: https://junit.org/junit4/faq.html#overview_1 [Accessed 04 05 2019].

[2] "TestNG Documentation," [Online]. Available: https://testng.org/doc/documentation-main.html [Accessed 04 05 2019].

[3] "Mockito framework site," [Online]. Available: https://site.mockito.org/ [Accessed 04 05 2019].

[4] Poirier, Y. "What are the Most Popular Libraries Java Developers Use? Based on Github's Top Projects," [Online]. Available: https://blogs.oracle.com/java/top-java-libraries-on-github [Accessed 04 05 2019].

[5] Dutheil, B. "Mockito vs EasyMock," [Online]. Available: https://github.com/mockito/mockito/wiki/Mockito-vs-EasyMock [Accessed 04 05 2019].

[6] "Software Testing – 2019/19 Spring - Labs," [Online]. Available: https://courses.cs.ut.ee/2019/SWT2019/spring/Main/LabsPracticeSessions [Accessed 08 05 2019]

[7] "Mock | Definition of Mock by Merriam-Webster," [Online]. Available: https://www.merriam-webster.com/dictionary/mock [Accessed 08 05 2019]

[8] Paralogarajah, P. "What is Mocking in Testing?" [Online]. Available: https://medium.com/@piraveenaparalogarajah/what-is-mocking-in-testing-d4b0f2dbe20a [Accessed 08 05 2019]

[9] "What is an Integration Testing?" [Online]. Available: http://www.professionalqa.com/integration-testing [Accessed 04 05 2019]

Appendix

I. Lab Materials

Lab Materials for Students and Lab Supervisors

- Lab instructions “Integration testing”, PDF file
  https://www.dropbox.com/s/uqbepet7xgzg71d/SWT2019-lab06.pdf?dl=0

Operator Communicator Source Code

- Operator Communicator Java Project – Lab Version (version intended for testing in lab), ZIP file

  https://courses.cs.ut.ee/2019/SWT2019/spring/uploads/Main/operator-communicator-student-version.zip

- Operator Communicator Java Project – Homework Version (version intended for homework), ZIP file
  https://www.dropbox.com/s/jwbpba3pz5jm11q/Integration%20Testing%20Homework%20%28Student%20Version%29.zip?dl=0

- Operator Communicator Java Project – Lab Version (full test coverage version for lab supervisors), ZIP file

- Operator Communicator Java Project – Homework Version (full test coverage version for lab supervisors), ZIP file

For confidentiality reasons, the lab supervisor versions are not public, but can be requested from the author.

II. Student Feedback Form

Feedback about Lab 6 – Automated Integration Testing / Mocking

Name: _____ (optional)

Scale:
-2 => strongly disagree
-1 => disagree
0 => so-s0
+1 => agree
+2 => strongly agree

	check exactly one box				
	-2	-1	0	+1	+2
Q1: The **goals** of the lab were clearly defined and communicated					
Q2: The **tasks** of the lab were clearly defined and communicated					
Q3: The **instructions** of the lab were appropriate and helpful					
Q4: The **tools** used in the lab were appropriate and useful					
Q5: Compared to the previous labs, the in-class exercises were more difficult					
Q6: Overall, what I learned in the lab session is relevant for working in the software industry					
Q7: Overall, the lab was interesting and inspiring					

Here you can add additional feedback:

III. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Mari-Liis Pihlapuu**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Software Testing: Integration Testing Lab Package,

supervised by Dietmar Pfahl.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Mari-Liis Pihlapuu

08/05/2019