UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Andri Jasinski

Lab Package: Automated Testing Using CI/CD

Bachelor's Thesis (9 ECTS)

Supervisor: Dietmar Pfahl, PhD

Tartu 2019

Lab Package: Automated Testing Using CI/CD

Abstract:
The primary goal of this bachelor's thesis is to create a lab package about automated testing using a continuous integration and delivery system for the course Software Testing (LTAT.05.006) at the University of Tartu. The thesis introduces the materials produced for this lab, analyses the feedback gathered from students and makes suggestions for future improvements.

Keywords:
Software testing, lab package, Selenium, Gherkin, Python, web application

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

Praktikumimaterjal: Automaattestimine kasutades CI/CD-süsteemi

Lühikokkuvõte:
Antud bakalaureusetöö põhieesmärgiks on praktikumimaterjalide loomine automaattestimise kohta kasutades pideva integratsiooni ja tarne süsteemi aine "Tarkvara testimine (LTAT.05.006)"jaoks, mida loetakse Tartu Ülikoolis. Töös kirjeldatakse loodud materjale, analüüsitakse tudengite tagasisidet ja tehakse ettepanekuid edaspidiseks arenduseks.

Võtmesõnad:
Tarkvara testimine, praktikumimaterjal, Selenium, Gherkin, Python, veebirakendus

CERCS:
P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

1 Introduction

Software testing is a continuous process of investigating and asserting software or system quality before releasing it to the end users. It is achieved by applying various software testing techniques and some of them are introduced in the Software Testing course at the University of Tartu. The course gives second-year Computer Science undergraduate students the basic knowledge of software testing via lectures, lab sessions and homework.

The labs are intended for the students taking the Software Testing course (mainly second-year Computer Science bachelor's degree students). There are 11 labs in total. Each lab is designed to give an introduction to a new software testing technique aimed to localize faults in the software. In the past years, behaviour testing has not been covered in the context of web application testing and continuous integration & delivery systems have not been observed in any lab or lecture.

The purpose of the thesis at hand is to create a full package of materials for the new lab session titled "Web-Application Testing in the CI/CD Pipeline", containing a system to be tested (web application), lab instructions, lab supervisor's instructions, a grading scheme, in-class and homework tasks. The main purpose of the lab is to teach students to write acceptance tests using the following technology stack: Gherkin programming language, Behave package (Python's Cucumber implementation) and Django web framework. Students should be able to create a test suite that would find 11 bugs in the system under test. Bugs are injected intentionally, they are disjointed and students are provided with their approximate location. The second purpose of the lab is to give students an introduction to continuous integration and delivery systems. As for the continuous integration part, students are asked to use the Bitbucket Pipelines CI/CD system and commit their changes as often as possible. To understand continuous delivery, students will have to set up deployment of the web application to Heroku web server. They are provided with the Bitbucket Pipelines configuration file. It is configured to run tests as the first step and deploy to Heroku as the second step if the testing step is passed successfully. As part of the thesis, the lab package was created, evaluated in the lab and feedback about it was collected and analysed.

This thesis contains four main sections in addition to "Introduction" and "Summary and Conclusions". The second section provides an overview of the Software Testing course and the existing materials and gives an introduction to the Gherkin programming language and the Cucumber framework that uses Gherkin as well as to agile methodologies and behaviour-driven development. The continuous integration and delivery practices are also described in this section. The third section describes the lab package design — schedule, materials, pet clinic web application and more. The fourth section describes lab execution observations and the fifth section consists of the description of the feedback collection process, students' feedback in graphs and feedback analysis.

2 Background Information

The following subsections give an overview of the Software Testing course and the main topics of the lab package.

2.1 Course Software Testing

At the University of Tartu, Software Testing is a 6 ECTS course. It is taken by Computer Science students of the bachelor's programme in the fourth semester. In addition to Computer Science students, this course can be also taken by other students who have passed a set of compulsory prerequisite subjects [1]. As the course outline states [1], it covers the foundations of software quality assurance by introducing different testing strategies. The course consists of 11 labs and the following list contains the topics of each lab:

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 Behaviour Testing

Behaviour testing is a black-box testing technique that asserts software's external behaviour by running manually written test cases. It has many implementations, but the most widely used ones are Gherkin-based implementations.

2.2.1 Gherkin

Gherkin allows stakeholders and developers to co-operate on the project by creating acceptance test cases and mocking the system use [3]. It is mostly used to document the desktop application and/or cloud service in a natural language without detailing how software behaviour is implemented [3]. It can be achieved by writing a set of user-stories-like "scenarios" around specific features. Those scenarios are stored in Gherkin files with `.feature` extensions using a specific set of keywords — `Feature`, `Scenario`, `Given`, `When`, `Then`, `And`, etc [3]. Every Gherkin file starts with the `Feature` keyword containing the name of the feature and an optional description. It is followed by one or many scenarios that are sharing the same template. Scenarios contain (but may not) the context, the event and the outcome [3] — the `Given`, `When`, `Then` keywords accordingly.

To give a first-hand example, imagine a system that should allow the user to login. It should handle basic use cases like logging in with valid and invalid credentials. If logging in succeeds, the user should be redirected to the home page, and if not, user should be notified with an error message and be able to log in again. Let us construct two user stories based on this specification:

1. As the user, I want to be redirected to the home page on a successful login so that I can see the content straight away.
2. As the user, I want to be notified with an error message if I have typed in a wrong username/password so that I can try to log in again.

Since these user stories are stating preconditions, actions and business values, it is possible to create the Gherkin file shown in Listing 1:

```
Feature: Logging in

  Scenario: Login with valid credentials

    Given an anonymous user
    When user submits valid login credentials
    Then user is redirected to / page # Home page

  Scenario: Login with invalid credentials

    Given an anonymous user
    When user submits invalid login credentials
    Then user is notified with login failed message
```

Listing 1. Gherkin file example

The feature under test is now described better and it is easier to understand the behavior of the functionality in different situations thanks to the steps definitions. Any business person should be able to come up with a simple feature file using this example. However, the real-world acceptance test engineering process does not stop here.

2.2.2 Cucumber

Since Gherkin is just a language, it has to be used by some framework. In this case, Gherkin is mostly used along with Cucumber which is an open-source tool for running acceptance tests. It was originally used by Ruby developers to create the logic behind the scenario steps. Its first stable release (version 1.0.0) was published on 21st June 2011 [4]. After that, Cucumber implementation was ported for Java, JavaScript, C++, Python and more programming languages.

The application areas of Gherkin and Cucumber are wide. Acceptance tests can be applied for command-line tools, desktop software, web and mobile applications. Since Cucumber was engineered with ease of use in mind, it allows all team members with different technical knowledge to contribute equally to product quality assurance [5]. This allows teams to see acceptance tests not only as a part of the testing process in agile methodology [6], but rather a colloboration tool for everyone involved in the development of a piece of software [5].

Let us assume that a software engineering team including developers and QA specialists had a Gherkin file from Listing 1 as well as initial user stories. To create an acceptance test out of that, a developer has to tie each step in the scenario to the function that will execute an appropriate command for the step logic. Since Cucumber has support for many programming languages, let us implement tests in Python.

```
from behave import given, when, then
from app.test.factories.user import UserFactory

@given('an anonymous user')
def step_impl(context):
    u = UserFactory(username='foo', email='foo@example.com')
    u.set_password('bar')
    u.save()


@when('user submits valid login credentials')
def step_impl(context):
    br = context.browser
    br.get(context.base_url + '/login/')

    br.find_element_by_name('username').send_keys('foo')
    br.find_element_by_name('password').send_keys('bar')
    br.find_element_by_name('submit').click()
```

```
@when('user submits invalid login credentials')
def step_impl(context):
    br = context.browser
    br.get(context.base_url + '/login/')

    br.find_element_by_name('username').send_keys('foo')
    br.find_element_by_name('password').send_keys('bar is invalid')
    br.find_element_by_name('submit').click()


@then('user is redirected to {url} page')
def step_impl(context, url):
    br = context.browser
    assert br.current_url.endswith(url), \
      'User is redirected to a wrong url'


@then('user is notified with login failed message')
def step_impl(context):
    br = context.browser

    assert br.current_url.endswith('/login/')
    assert 'Please enter a correct username and password'\
      in br.page_source, 'Error message is not visible on page'
```

Listing 2. Python implementation of Gherkin file example using Behave framework

In the Python world, Cucumber implementation is called Behave[1]. Backed by the versatility of Python, it makes Behave a very powerful tool. As shown in the Listing 2, tests are performing many actions, such as database operations, elements look-up in the HTML DOM and pattern matching.

2.2.3 Agile Methodology and Behaviour-Driven Development

The agile software development methodology is a well-known set of rules and processes in the modern software developer community. The core methods behind agile development were designed and used back in 1957 and only have evolved with time [7]. The first round of interest in iterative and incremental development (IID) methods was in the 1970s [7]. It was a really promising replacement for the waterfall model, and by 1990s, IID was already loved by computer science students and was slowly moving to the corporate segment [7]. In 2001, a group of seventeen developers gathered together to compose a new set of development best practices. Later, they published the "Manifesto for Agile Software Development" [8]:

[1]Behave `https://behave.readthedocs.io/en/latest`

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

They have also published "Twelve Principles of Agile Software" [8]. Here are some principles that are related to software testing and quality assurance:

- Business people and developers must work together daily throughout the project.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.

This is showing that Cucumber suits the agile software development methodology well. As mentioned above, it allows all team members with different technical knowledge and different responsibilities to be on the same page regarding technical and non-technical requirements throughout the process of software development.

With time, the benefits of agile methodologies over the waterfall and many other practices created interest in the safety-demanding segment [9]. When developing such software following agile development rules, it was a serious challenge to ensure that shipped software meets security requirements and passes all verifications. That was the moment when the processes of test-driven development (TDD) and behavior-driven development (BDD) were introduced as part of agile software development. Test-driven development is all about covering the code with tests even before it is written. The TDD approach can be simplified down to three points [10]:

1. Write automated test(s) that will fail at the beginning
2. Write code that implements the expected functionality and makes test pass
3. Refactor the code

Behaviour-driven development follows the same principles as TDD. However, it gives all team members, not only developers, the oportunity to collaborate on the product quality assurance even before the first line of code is written. It can be done already when functional and non-functional requirements are gathered and the user stories are created. BDD covers testing of different kinds of software — from low to high-level. Because Cucumber allows writing acceptance tests not only for the graphical user interface but also for REST APIs [5], command-line interfaces and more, it was a great match for

BDD. Over the years, more and more teams started to introduce TDD and BDD into their software development process and used acceptance tests to assure the quality of the frontend by interacting with HTML elements using the Selenium tool, or RESTful backend by sending HTTP requests and asserting the expected response to the received one [5].

2.3 CI/CD

Continuous integration (CI) and continuous delivery (CD) systems are known to the agile software development community for a long time and have been evolving since 1980s [12]. However, the benefits of CI/CD were overlooked by researchers all over the world [11]. The following subsections will give an overview of various CI/CD paradigms, its benefits and usages in the modern world.

2.3.1 Continuous Integration

Continuous integration is not a tool — it is a practice, a methodology in software development automation. CI is based on the idea of regularly integrating code changes into the existing code base and running the tests [12]. The phrase "continuous integration" was introduced by Grady Booch in 1991 in his book "Object-Oriented Analysis and Design with Applications" and was often used in Extreme Programming publications in the late 1990s [12]. From then on, CI started to conquer the community (especially the agile software development community) and began to be used more often in enterprise corporations. It was a big step further for the industry. For example, HP has reported that continuous integration helped them reduce their development costs by 78% [11]. In 2018, JetBrains published their "The State of Developer Ecosystem"[2] survey which states that 44% of 6000 surveyed developers are using a continuous integration and/or continuous delivery system.

In 2006, Martin Fowler [17] came up with the list of preferred practices for continuous integration implementation [12]. The following list contains the CI practices, a summary of every point [17] and thesis author's thoughts about whether it is applicable for modern software engineering processes:

- *Maintain a Single Source Repository*

  By "Single Source Repository" Fowler means a version control system's repository (Git, CVS) that contains everything needed to set up the build environment — from build scripts to third-party libraries and compilers. Nowadays, it is not so important due to Internet speed improvements and dramatically increased size of dependencies. The current trend is to have there only source code and some configuration files/scripts to download third-party libraries or initialise the Docker container, for example.

- *Make Your Build Self-Testing*

  This is not strictly a requirement for continuous integration but rather a recommendation for software architecture. It is very important to cover the code with tests,

[2] "The State of Developer Ecosystem in 2018" by JetBrains. `https://www.jetbrains.com/research/devecosystem-2018/`

and Fowler as a TDD evangelist recommends developers to use TDD together with CI, stating the following point of view: "Imperfect tests, run frequently, are much better than perfect tests that are never written at all" [17].

- *Automate the Build*

  In this section, Fowler again emphasizes the point of view he has stated in the *Maintain a Single Source Repository* section description — one should be able to build the application from the source code on a fresh and clean machine by executing a single command. It is a very useful recommendation not only for continuous integration but also for the development process in general because setting up the development environment from scratch for an existing project is always a hassle.

- *Everyone Commits To the Mainline Every Day*

  As the headline states, it is important that developers keep their progress in sync. This makes it easier to resolve conflicts when changes are committed every few hours or even more often. However, Martin Fowler seems to speak about the `master` branch of the repository. The author of the thesis wants to emphasize that for a project in production state it is important to open `feature` branches from the master one in order to keep the not-ready code away from the latest and stable version of the service. Once the feature is ready to be released, the `feature` branch has to be merged into the `master` branch.

- *Every Commit Should Build the Mainline on an Integration Machine*

  Since developers are most likely using different machines, the build may work on one machine, but may not work on the other. With that said, application builds that are running on the integration machine with the correct environment have a higher possibility to be fully operational if the source code does not contain errors. It is also the responsibility of the developer as the commit author to monitor the build and fix it if fails.

- *Fix Broken Builds Immediately*

  In this section, Martin Fowler again seems to be speaking about the situation of committing straight to the main branch. The impact to the business can be significantly reduced by developing in a separate branch and syncing with the main branch at least once per day.

- *Keep the Build Fast*

  There is no real reason to have 1-hour builds. A long build means a higher power draw during the build in the case of self-hosted CI system or `N`-amount of time reduced from the time credit of a cloud CI system. This is why a group of

developers came up with the "Ten-minute build" rule [12]. It states that build and test steps should pass within 10 minutes. Having a fast build is very important so that developers can communicate within a few minutes after the faulty code was added to the repository and discovered. Nowadays, build time can be improved by using a faster build machine, having heavy-weight dependencies pre-installed in the environment, using isolated systems, such as Docker, and optimizing the build configuration (changing the compiler, etc). Martin Fowler also suggests that if the build takes too long, it is important to divide it into steps — the first step should be the most important one that will deliver a notification to the development team if the pipeline has failed. The next steps can be testing (also separated into steps or even run in parallel) and deployment.

However, not all platforms are able to meet the ten-minute build rule, one example is mobile application building. The compilation alone can take up to 20 minutes, and if tests are included, we are looking at approximately an 1-hour build. There is definitely room for improvement and the mobile app development community is not standing still.

- *Test in a Clone of the Production Environment*

  This requirement is not always easy to reach. Depending on the application type, the production environment can be changeable as in the case of desktop applications. It is not possible to test all the possible desktop environments people will be running the application in, but developers have come up with ways to build apps at least against different operating systems. One of the examples is the Travis-CI system that allows to run `matrix` jobs (in parallel) with different OS'es[3]. If your application is web-based, a cloud service or again a desktop app but for tech-savvy users, you can go with Docker since it allows you to develop, run tests, production environment and continuous integration system with the same environment configuration and also distribute pre-built containers to all users interested in it.

- *Make it Easy for Anyone to Get the Latest Executable*

  This is the place where continuous delivery comes into play — it is very important for users to be able to download and run the latest and stable executable. Everyone is free to decide where they want to store it and how, but manual distribution should not be the developer's concern.

- *Everyone Can See What's Happening*

[3] "Testing Your Project on Multiple Operating Systems" by Travis-CI. `https://docs.travis-ci.com/user/multi-os`

This section explains the importance of the dashboard that will mirror the builds history and give a fast and clear overview of what was the state of the last build. It is still very useful even today because that way developers, business people and quality assurance engineers are on the same page with regard to product health. As we already know, communication is highly important for agile methodologies.

- *Automate Deployment*

  The final section speaks about the benefits of continuous deployment, describes the possible problems and discusses how to resolve them. Continuous deployment is described in details in the next section.

2.3.2 Continuous Delivery and Deployment

Continuous delivery comes side by side with continuous integration. When the testing step is passed and the new version of the application is built, the next question arises — how to deliver the new version of the web, mobile or desktop application? Continuous delivery systems are answering this and many more questions.

The purpose of continuous delivery systems is to speed up and ease the process of supplying end user with the latest build of the service (deployment to web server or delivering the executable binary), lower the risk of faulty build due to the difference in the environment and increase productivity by delegating heavy load and time-consuming tasks from developers' machines [13].

Studies show that companies can benefit from introducing continuous delivery into their release pipeline. The benefits include, but not limited to, increased and more stable collaboration between developers and customers, improved productivity, quality and decreased risk of failures [13]. To give a real-world example, in 2018, the CI/CD tool for mobile developers Nevercode claimed in their blog post that they have saved 20% of a developer's work day just by combining continuous integration and delivery for the project[4]. Taking into account that mobile application build and distribution time can be bigger than, for example, Docker container build time, the respective number for general-purpose CI systems can be higher than 20%.

However, the continuous delivery and deployment pipeline is not always 100% stable. It is highly important to have a plan B — ability to roll back any faulty deployment and return the system to an operating state [17]. Nobody is insured from deploying a buggy build that is partially or completely breaking the application. Redeployment can be done by special tools or by reverting a faulty commit. Anyone from the development team should be able to do so. Preferably, from any available device.

[4] "What we learned about CI/CD analysing 75k builds" by Nevercode. `https://nevercode.io/blog/what-we-learned-about-ci-cd-tool-analysing-75k-builds`

3 Lab Design

The following subsections give an overview of the materials created for the "Web-Application Testing in the CI/CD Pipeline" lab session.

3.1 Lab Schedule

As the "Software Testing" course outline states, the amount of hours expected to be spent on independent work (outside classroom) is 92 person-hours [1]. The homework task should take approximately 5 person-hours to solve [2]. Students are working in pairs. The "Web-Application Testing in the CI/CD Pipeline" lab schedule is as follows:

- Approx. 70 minutes for development environment preparation, introduction to behaviour testing and continuous integration and delivery, writing the first test case with the help of the supervisor.
- Approx. 20 minutes to overview the homework task and answer questions.

3.2 Lab Materials

The author of the thesis created the following materials for the lab "Web-Application Testing in the CI/CD Pipeline":

- Lab instructions for students containing a specification of the system under test, the local environment setup tutorial, the continuous integration and delivery system guide and the example of a test case for the login page.
- Lab supervisor's instructions containing everything from lab instructions for students and links to test cases completely covering the system under test, an example of test cases for the "Register" page and the list of bugs that the system under test contains.
- The Pet Clinic web application that is available in two different versions:
  - The "master" [14] version that is clean of bugs.
  - The "lab" version that contains 11 bugs on different pages.

Links to lab materials can be found in Appendix I. In addition to setup instructions and two test examples (in the case of the lab supervisor version), lab materials contain an introduction to lab, homework description and grading schema.

3.3 Pet Clinic Application

The Pet Clinic "Bark!" web application is a cloud service that allows pet owners to book the visit to the veterinarian. This service was created as part of this thesis in order to act as a system under test for students of the Software Testing course. Besides booking the visit, the web applications allows users to register, log in, add pets, browse the list of vets, manage booked visits by changing the vet, visit time or date and modify personal information.

Since the author of the thesis is a Python developer, the technologies stack was easy to predict — Python as the main programming language using the Django web framework. It was chosen because it is easy to set up from scratch, has good integration with SQL databases (common for second-year undergraduate students) and template rendering engine. The web user interface was developed using Bootstrap.

Students receive the faulty version of the Pet Clinic "Bark!" application. It contains 11 bugs that are disjoined from each other, hence are not blocking, and students can decide in which order they are going to test the cloud service. Bugs are located at:

- Profile page — `/profile`
- Vets page — `/vets`
- Visit booking form

This information is given to students along with the service specification.

3.4 Lab Session Tasks

The lab session tasks have the following purposes:

- Guide students through the process of setting up the development environment for an existing project to simulate the process of onboarding at the new job
- Introduce students new technologies, such as Django, Gherkin and cloud hosting services on the example of Heroku
- Give an introduction to the homework task

The lab should start with the introduction to behaviour testing by lab supervisor. After that, lab supervisor can show the "master" version [14] of the web application to students and tell about the features of the application. All features are also described in lab instructions under the "Specification" section.

After students have a clear understanding of the system under test, they can start to set up the local development environment. To do so, they need to download Git and

Python (if they do not have it installed before the lab session) and application source code. The code they will receive already contains the 11 bugs described above.

When students download the source code and install third-party dependencies, they will need to migrate the database schema and, optionally, create a superuser and populate the database with a sample set of veterinarians. To ensure that the setup is done correctly, students are asked to run the web application on `localhost`.

The next step is to get first-hand experience with Gherkin by executing their first test case. They are provided with the login page tests containing two scenarios — one for logging in with valid credentials and one for logging in with invalid credentials. It is a small and good example because it demonstrates how the Gherkin application works (testing one software feature by stressing it under different scenarios), useful Gherkin features like steps reusability [15] and pattern matching [16] and an example of Python implementation. Students can conveniently copy the source code and place it in the appropriate location mentioned in lab instructions. Before executing login page tests, students need to download a webdriver. The system under test is configured to use `geckodriver` by Mozilla Firefox[5] and `chromedriver` by Chromium team[6]. If everything is set up correctly, students can execute tests locally and tests should pass successfully.

3.5 Homework Tasks

The homework assignment contains the following tasks:

1. Cover the cloud service with acceptance tests by creating a set of test cases using Gherkin and the behave-django framework.

2. Familiarize with the continuous integration and continuous delivery tool Bitbucket Pipelines.

Task 1 expects students to do a lot of manual research — they will need to go through the documentation for different technologies, such as Gherkin, behave, behave-django, django QuerySet API, Selenium, etc. The purpose of that is to show students the real-world example of having to work with technology they may not be familiar with at all.

Since the application contains 11 bugs, we expect students to have at least 11 test cases. Test cases for login and register pages written during the lab sessions are excluded. Students are also provided with the explanation of a "done test case":

- The test case is in appropriate feature file

[5] geckodriver by Mozilla. `https://github.com/mozilla/geckodriver`

[6] chromedriver by Chromium. `http://chromedriver.chromium.org`

- The logic behind scenario steps is implemented in Python and placed in the *features/steps* folder
- The test case is passing or it is failing due to an encountered bug

Task 2 expects students to get the understanding of basic usage of continuous integration and delivery systems in general. Students are asked to commit every finished test case. This will simulate the process of developing tests for the service that already contains some functionality. CI/CD is used in this practice session because it is not covered in any course, however, CI/CD implementation is required in the Software Project course at the University of Tartu. When taking the Software Project course, the author of the thesis was not aware of CI/CD in any form and this is how the idea for this bachelor's thesis was born. The author hopes that the lab materials and introductions to behaviour testing solutions and CI/CD given in this thesis will provide enough explanation about general purpose continuous integration systems so that students can make use of it during the Software Project course and in their future projects.

3.6 Grading

The lab session is measured at 10 points maximum. The grading schema is divided into four sections:

- 1 point for lab attendance
- 1.3 points for the usage of CI/CD system
- Maximum 4.4 points for 11 test cases (login and register pages tests are not included)
- Maximum 3.3 points for 11 discovered bugs

Test cases for the login and register pages are not included in the 11 test cases. If a student has less than 11 test cases, he/she will receive only 0.4 points times the number of test cases. If a student finds less than 11 bugs, he/she will receive only 0.3 points times the number of found bugs. If a student writes more than 11 test cases, it is still treated as 11.

4 Lab Execution

The Software Engineering course had eleven labs in the spring semester of 2019. The "Web-Application Testing in the CI/CD Pipeline" was the seventh lab in a row and was given to 4 groups on 2nd and 3rd of April.

The basics of behaviour testing were explained to students in the lecture on the 21st of March. Students who attended the lecture should be familiar with Gherkin and Cucumber. The author of the thesis was present in 3 lab sessions out of 4 to observe the time consumption of each lab section, possible points for improvements and to react to any possible issues during the lab execution.

During all sessions, lab assistants started with the introduction of the core components of the lab — behaviour testing, the system under test and CI/CD. During the first lab session, Windows users encountered a problem with Python's third-party dependencies installation. To resolve this, they had to download the "Microsoft Visual C++ Build Tools". After this lab, the author of the thesis created a virtual machine (VM) running Ubuntu 18.04.2 long-term support version (LTS). All students were aware of the VirtualBox tool and were able to set up the environment and start developing quickly. The Ubuntu virtual machine had only Python 3 since Ubuntu comes with Python 3 preinstalled. The other tools, such as Git and Heroku CLI, had to be installed manually. This way, students using the VM had to go through the same installation process as the students working on their local machine.

Once this problem was overcome, students proceeded with the required dependencies installation and execution of the given example test case. As soon as the local development environment and CI/CD using Bitbucket Pipelines were configured, some students had enough time to proceed with in-class exercise — writing test cases for the `Registration` page.

The lab session took roughly 90 minutes and some students stayed in the class to ask questions from the lab assistant or the author of the thesis. At first sight, students found this lab package interesting. They tended to ask for help and were looking forward to know more about Gherkin and its features. However, the author of the thesis expected students to be more proficient with Git since Software Engineering (SE) is a prerequisite for the Software Testing course and Git is highly covered in SE.

5 Feedback

The following sections include an overview of the feedback received from students and its analysis. This also includes a section about the author's suggestions on how to improve the existing lab based on the lab execution experience and gathered feedback.

5.1 Feedback from Students

One week after lab execution, students received the questionnaire with 7 statements and an empty field to write additional feedback in. The statements were as follows:

1. The goals of the lab were clearly defined and communicated.
2. The tasks of the lab were clearly defined and communicated.
3. The instructions of the lab were appropriate and helpful.
4. The tools used in the lab were appropriate and useful.
5. Compared to the previous labs, the homework assignment was more difficult.
6. Overall, what I learned in the lab is relevant for working in the software industry.
7. Overall, the lab was interesting and inspiring.

The copy of the questionnaire can be found in Appendix II. The questions are aiming to find out whether students found the lab instructions, tools and tasks appropriate and helpful, how complex the lab was compared to previous assignments and if it was interesting and inspiring. Students had five answer options:

- -2 — strongly disagree
- -1 — disagree
- 0 — so-so
- 1 — agree
- 2 — strongly agree

Feedback was received from 58 students which is 59% of all the students registered to the Software Testing course (98 students in total). The following figures show the distribution of the students' opinion on all seven questions.

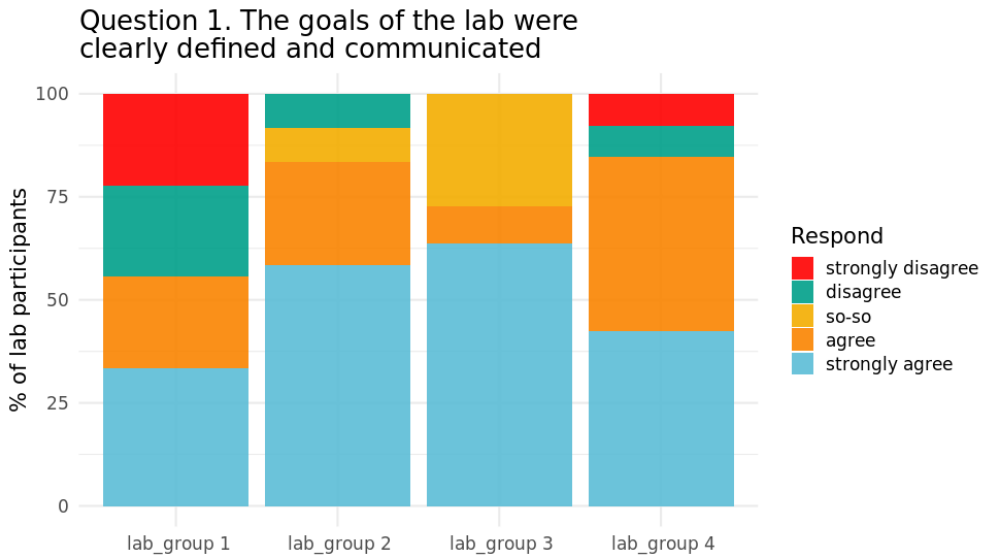

Figure 1. Feedback to "The goals of the lab were clearly defined and communicated"

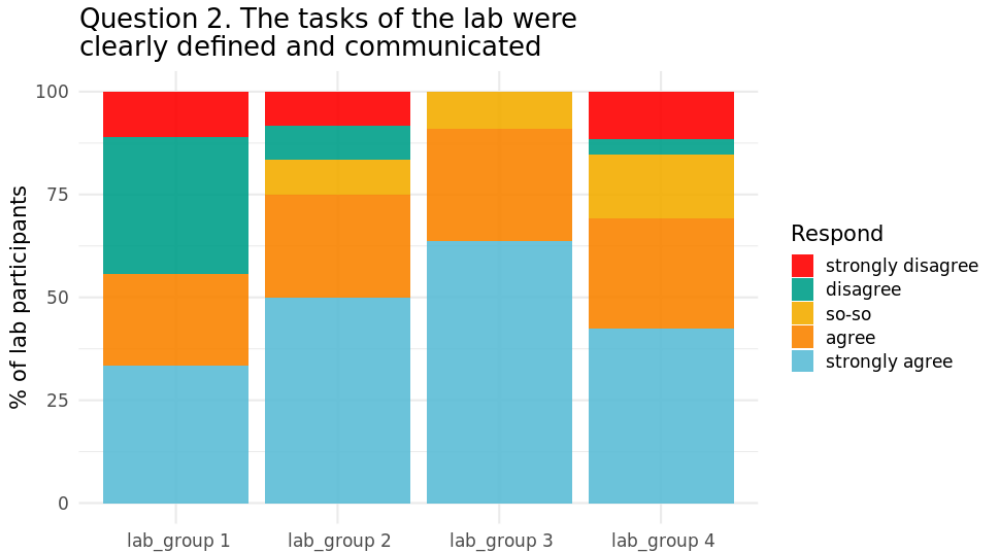

Figure 2. Feedback to "The tasks of the lab were clearly defined and communicated"

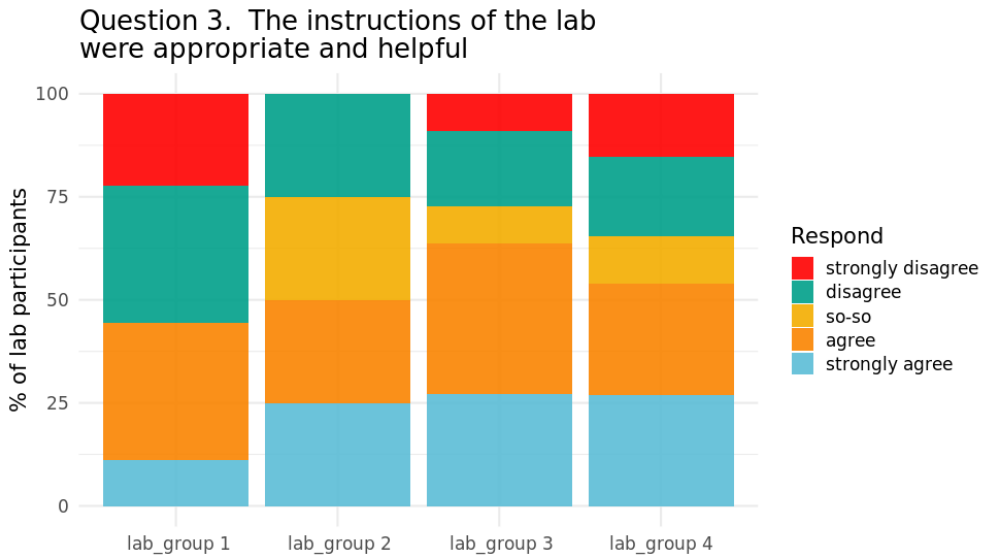

Figure 3. Feedback to "The instructions of the lab were appropriate and helpful"

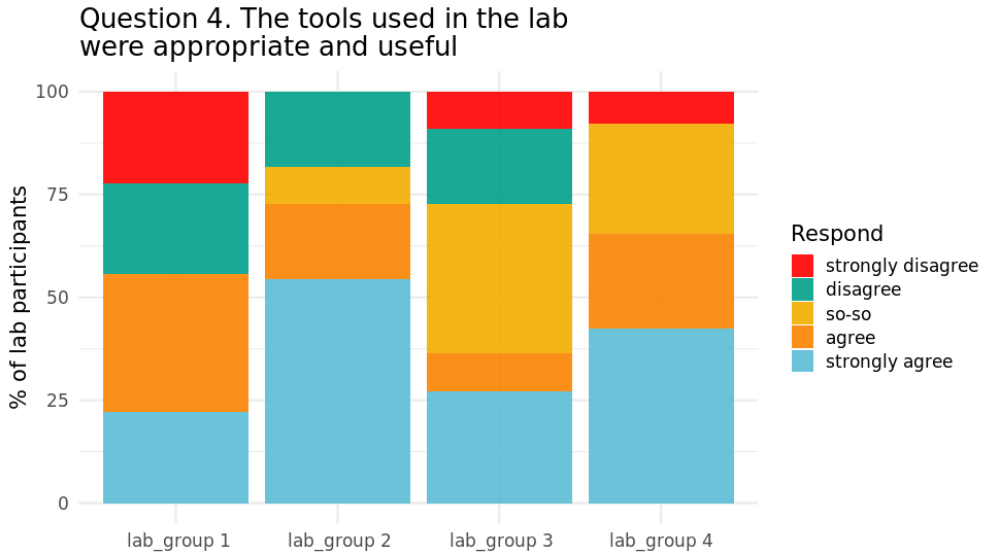

Figure 4. Feedback to "The tools used in the lab were appropriate and useful"

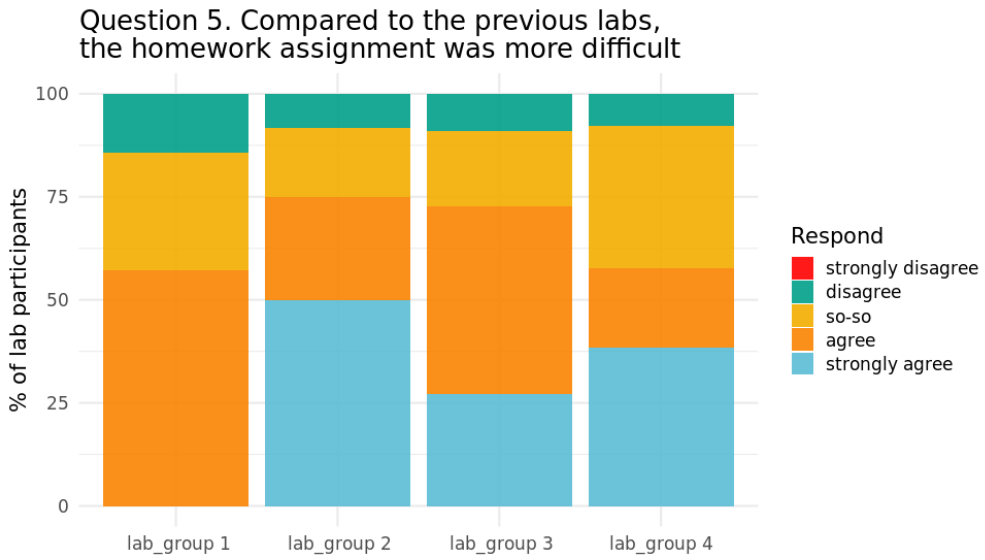

Figure 5. Feedback to "Compared to the previous labs, the homework assignment was more difficult"

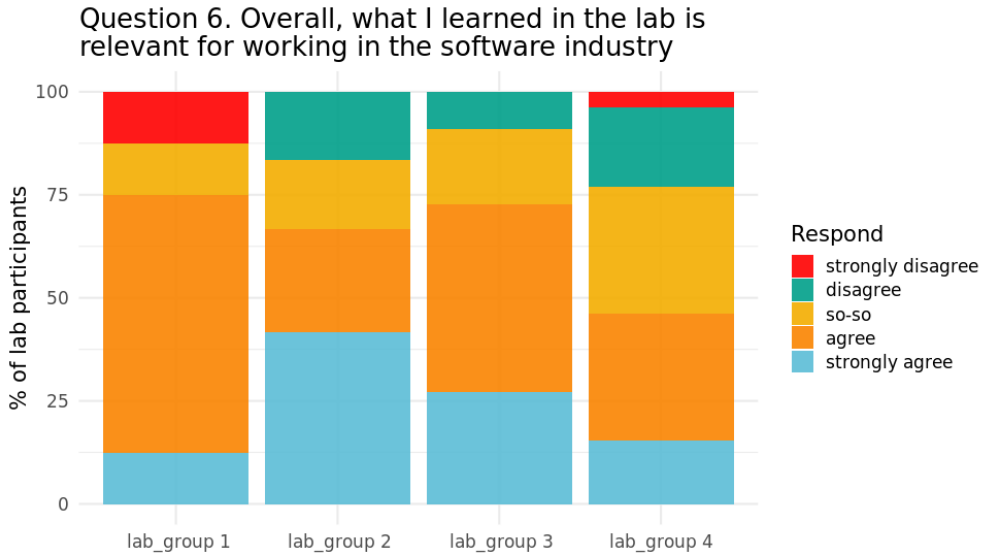

Figure 6. Feedback to "Overall, what I learned in the lab is relevant for working in the software industry"

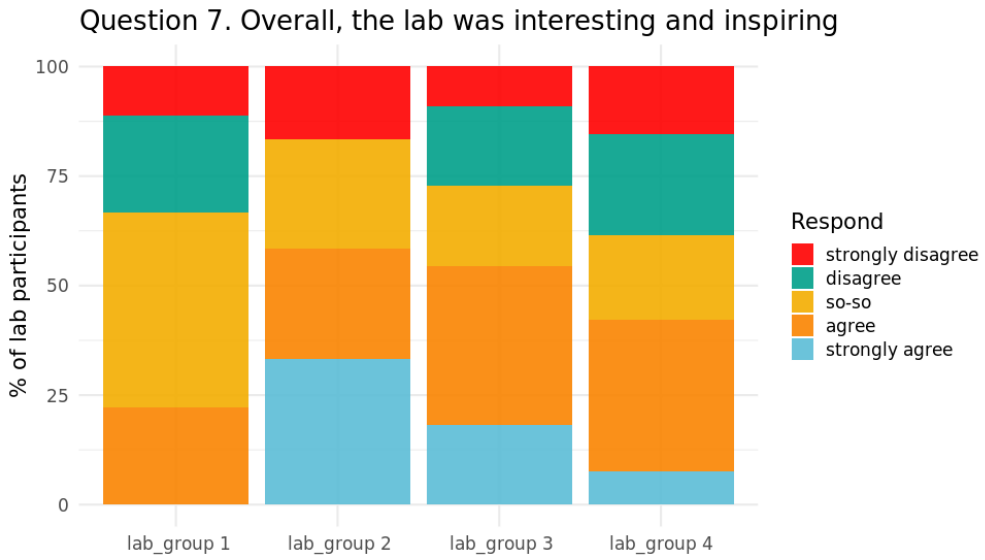

Figure 7. Feedback of "Overall, the lab was interesting and inspiring"

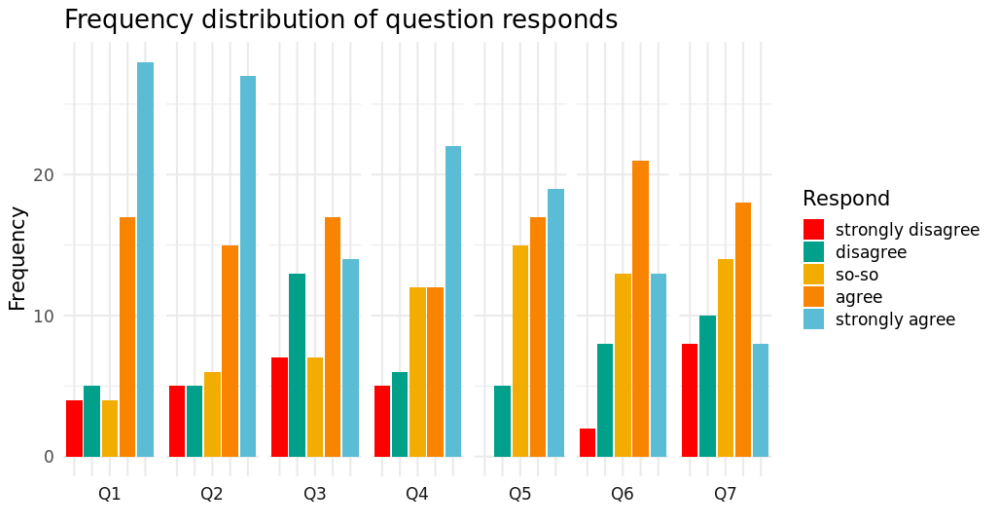

Figure 8. Summary for all questions including all lab groups

5.2 Analysis

The figure 1 shows that students mostly understood the main goals of the lab. Only 2 students in the lab group 1 and in the lab group 2 strongly disagreed with the statement.

Figure 2 also shows mostly positive feedback for the second question "The tasks of the lab were clearly defined and communicated".

Regarding the lab instructions, students tend to be rather neutral. In figure 3, it is visible that students in the lab group 1 have a stronger opinion (the "so-so" option was not marked by any student), but the overall majority answered neutrally or positively.

Students have mixed feelings regarding the tools used in the lab. For example, over 50% of the students in lab group 2 liked the tools (marked "strongly agree" option), however, nearly 50% of the students in lab group 1 responded negatively to this statement (figure 4). One student from lab group 4 expressed his/her discontent regarding the usage of Python.

As figure 5 shows, at least half of the students in each lab agreed that the homework assignment was more difficult than previous ones. It may or may not be treated as a positive outcome. The lab was intended to be more complex than the previous lab, but not too difficult.

Figure 6 is showing the neutrality or agreement of the students regarding the sixth statement — "Overall, what I learned in the lab is relevant for working in the software industry". The number of negative opinions is peaking at around 25% in the lab group 4, which is considered to be a good result.

Last but not least, students are mostly neutral or positive regarding the seventh question — "Overall, the lab was interesting and inspiring". It is highly important that students are motivated to complete the tasks even if the lab is more complex than what they have done before.

In the "additional feedback" section, some students expressed their discontent regarding problems they had faced with Windows machines. Fortunately, most of them managed to overcome the problems by using the virtual machine, but one of the student had reported that "it was not working". However, as lab supervisors have reported, no students contacted them regarding this issue.

Overall, the thesis author considers lab execution to be successful and the feedback analysis supports this statement. As expected, the first lab group had a stronger opinion as they were the first group to try out the lab materials and face issues. Other 3 groups are more neutral in their answers, however, it is clear that everyone's feedback is different. This is interesting within the confines of this one particular lab session.

5.3 Future Improvements

After the lab execution, the following list of possible improvements was composed taking into account both the students' and lab supervisors'[7] feedback:

1. Distribute the virtual machine running Ubuntu to students in advance. Make the virtual machine usage a prerequisite for this lab.

[7]The feedback from lab supervisors was gathered and discussed after each lab session.

2. Pre-install all dependencies and tools for the lab in order to reduce the development environment preparation time.

3. Give a more detailed overview of the Django web-framework and more examples of database queries and database usage.

4. Give more examples of Gherkin scenarios and its implementation using Behave.

5. Reduce the number of bugs in the web application.

The Software Testing course staff may or may not take these suggestions into consideration. The thesis author would definitely apply changes described in points 1 and 2. It would significantly decrease the amount of time spent on the installation routine and possibly increase the students' interest in this lab.

6 Summary and Conclusions

The goal of this thesis was to create a lab package containing a system to be tested, lab instructions, lab supervisor's instructions, the grading scheme, an in-class and a homework task. The lab package was created for the Software Testing (LTAT.05.006) course at the University of Tartu and given to 98 students registered to the course in the 2019 spring semester.

The lab execution went smoothly with some improvements applied after the first lab. 58 students provided feedback a week after the lab execution. Although the feedback was mostly positive, there is room for improvement. Based on the feedback of students and lab supervisors, the list of possible changes was created and shared with the course staff. All things considered, the created lab materials met course requirements and can be used in the future for the Software Testing course.

References

[1] "Course outline (2019)" [Online]. Available: https://www.is.ut.ee/rwservlet?oa_ainekava_info.rdf+1339229+PDF+6128386274228415539 2+application/pdf. [Accessed 05 03 2019].

[2] "Lecture 1 - Introduction to Software Testing" [Online]. Available: https://courses.cs.ut.ee/LTAT.05.006/2019_spring/uploads/Main/SWT2019_lecture01v03.pdf. [Accessed 05 03 2019].

[3] "A Natural Language Driven Approach for Automated Web API Development: Gherkin2OAS (2018)" [Online]. Available: https://doi.org/10.1145/3184558.3191654. [Accessed 07 03 2019].

[4] "Cucumber release list" [Online]. Available: https://github.com/cucumber/cucumber-ruby/releases. [Accessed 08 03 2019].

[5] Matt Wynne. 2012. *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf, Raleigh, NC.

[6] "What is Acceptance Testing in Agile" [Online]. Available: https://www.testingexcellence.com/acceptance-testing-agile. [Accessed 09 03 2019].

[7] "Iterative and Incremental Development: A Brief History" [Online]. Available: https://doi.org/10.1109/MC.2003.1204375. [Accessed 09 03 2019].

[8] "Manifesto for Agile Software Development" [Online]. Available: http://agilemanifesto.org. [Accessed 09 03 2019].

[9] "Combining STPA and BDD for safety analysis and verification in agile development (2018)" [Online]. Available: https://doi.org/10.1145/3183440.3194973. [Accessed 10 03 2019].

[10] Kent Beck. 2000. *Test-Driven Development By Example*. Addison-Wesley Professional, Boston, MA.

[11] "Understanding and improving continuous integration (2016)" [Online]. Available: https://doi.org/10.1145/2950290.2983952. [Accessed 12 03 2019].

[12] "Continuous Integration! You Keep Using Those Words. I Do Not Think They Mean What You Think They Mean. (2019)" [Online]. Available: https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-19-20745. [Accessed 12 03 2019].

[13] "Perceived Benefits of Adopting Continuous Delivery Practices. (2016)" [Online]. Available: https://doi.org/10.1145/2961111.2962627. [Accessed 12 03 2019].

[14] "Pet Clinic "Bark!"" [Online]. Available: https://bark-pet-clinic.herokuapp.com. [Accessed 17 03 2019].

[15] "Reusable Cucumber Steps" [Online]. Available: https://collectiveidea.com/blog/archives/2011/06/09/reusable-cucumber-steps. [Accessed 17 03 2019].

[16] "Cucumber Expressions" [Online]. Available: https://docs.cucumber.io/cucumber/cucumber-expressions. [Accessed 17 03 2019].

[17] "Continuous Integration" [Online]. Available: https://www.martinfowler.com/articles/continuousIntegration.html. [Accessed 17 03 2019].

Appendix

I. Lab Materials

Students Lab Materials

- Lab instructions "Lab 7: Web Application Testing in the CI/CD Pipeline", PDF file

  `https://courses.cs.ut.ee/LTAT.05.006/2019_spring/uploads/Main/SWT2019-lab07-20190402.pdf`

Lab Supervisor Materials

- Lab Supervisor instructions "Lab 7: Web Application Testing in the CI/CD Pipeline", PDF file

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

Pet Clinic "Bark!" Web Application Source Code

- Pet Clinic "Bark!" Web Application (version intended for testing), ZIP file

  `https://dyaz2zepqbyns.cloudfront.net/lab-package-full.zip`

- Pet Clinic "Bark!" Web Application (faultless version), ZIP file

For confidentiality reasons, source code of the faultless version is not made available in the thesis, but will be made available on request.

II. Feedback Questionnaire

Feedback to Lab 7 – Web-Application Testing in the CI/CD Pipeline

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 homework assignment was more difficult					
Q6: Overall, what I learned in the lab 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, **Andri Jasinski**,

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,

   Lab Package: Automated Testing Using CI/CD,

   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.

Andri Jasinski
06/05/2019