# COURSE INFORMATION

# TESTING AND CONTINUOUS INTEGRATION

## Introduction

In this course you will learn and apply different types of testing techniques by using a variety of JAVA frameworks. Also, best code practices & quality assurance will be discussed through a process of continuous integration.

## Prerequisites

* Knowledge of the JAVA language;
* Knowledge of Client-server architecture (CSA) and web responses (WEB1-2-3);
* Knowledge of a JAVA web application technique (RESTful or SOAP using either POJO and Apache Tom Cat or Spring);
* Math 3 (graphs theory and search algorithms);
* Knowledge of GIT (pulling, pushing, branching and merging);

## Learning goals

The learning outcomes of this course are:

* The student knows how to install a Jenkins continuous integration environment;
* The students can apply the use of Jenkins to a JAVA web application development;
* The students apply use of code styles in the assignment project;
* The students learn how to make their code decoupled by using building tools (Gradle) and manage dependencies;
* The students are capable of unit tests, using JUnit framework, that cover web application project they work on during the course;
* The students know how to write unit test that make use of mockups, using Mockito Framework, to cover the web application project they work on during the course;
* The student can make use of JMeter to run stress tests on his application and create a report with it (APDEX);
* The student learn about code coverage and perform code analysis making use of Intellj IDEA and applying it on the project they will work on.;
* The student practices collaboration by making pull requests through online collaboration tools and fixing them via GIT clients;

## Testing and final Assessment

### How is the subject tested?

During the lectures the student is prepared to face the practical assignments and receives formative feedback (to which extent did you achieve the learning goals) and feed-forward (constructive guidance on how to improve) from the teacher.

The final summative examination is conducted at the end and aims to verify the acquired skills. The test consists in a check that considers the learning goals, and is based on the group assignment.

The final mark in in the range 1 – 100 (translated then into the related two digits value). For a complete overview of the schedule, and a list of the assessment criteria, see description of the group assignment. Each individual student needs to be able to individually demonstrate, explain and be able to make changes to the handed in work.

### Additional Resources

All additional aids stored locally on your laptop are allowed to be used. Also web resources are allowed but plagiarism is **not** allowed. The code written by somebody else **must receive acknowledgment.**

### Retakes

The re-take is allowed in the following semester.

### Final Mark

Student gets a mark on the examination in the range 1 – 10. The value are rounded to half a point.

### Education hours

The course lasts 7 weeks. Each week there are 2 + 2 contact hours with the teacher. Some topics are shortly introduced by the teacher, but most studying needs to be done by the student himself. Topics can have reviews (quiz, class discussion, self-tests, etc.) which students can use to test their knowledge. Educational hours are also used for feedback and feed-forward.

For each topic a practical assignment is made available, which helps students to hone skills and for which students can get feedback from the teacher. These practical assignments are not mandatory but they represent a good tutorial to gain the skills needed for the final assignment.

### Support material

* Slides and practical assignments on the SharePoint.
* Book “Practical Unit Testing with JUnit and Mockito” by Tomasz Kaczanowski. (mandatory)

### Tutorials

* Each practical includes a part of suggested reading and tutorials.

# Planning

1. The following describes the topics covered each week and the corresponding book chapters suggested for reading.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. **W#** | 1. **class** |  | 1. **During week (individual)** | 1. **Practical (team of 3)** |  | 1. **Topics** |
| 1. **1** | 1. intro TCI |  | 1. study chapter 2. 1, 2, 3.1-3.5 3. do assignment 1 4. study chapter 3.6-3.10, 5.1 5. start assignment 2 | 1. **TCI install** |  | 1. Intro to course rules 2. Introduction to testing and continuous integration 3. Unit testing    1. JUnit    2. File Structure    3. Direct/Indirect inputs and outputs    4. Asserts 4. Jenkins intro 5. Gradle the building tool |
| 1. **2** | 1. review on week1 |  | 1. finish assignment 2 2. study chapter 5.2-5.3, 4 3. start assignment 3 | 1. **Introduction to TCI project** 2. **→** Set up of the project 3. → CI server setup 4. → Production server setup |  | 1. Unit testing    1. Parameterized Tests    2. Checking for thrown exceptions    3. Test fixture JUnit 2. Jenkins + Gradle, how to combine them 3. Gradle    1. Dependency management    2. Adding dep. to builds |
| 1. **3** | 1. review on week2 |  | 1. finish assignment 3 2. study chapter 5.4-5.6 3. start assignment 4 | 1. **TCI project start** |  | 1. Test doubles and types 2. Mocking, Stubs, Spy 3. Code coverage    1. Report |
| 1. **4** | 1. review on week3 |  | 1. finish assignment 4 2. study chapter 6.1-6.5, 7.7 3. start assignment 5 | 1. **TCI project continue** |  | 1. Unit testing techniques    1. Matchers and custom matchers    2. Testing the time    3. Capturing Arguments to Collaborators    4. Throwing exceptions 2. Pull requests 3. Reviews |
| 1. **5** | 1. review on week4 |  | 1. 8.4, 9.1-9.3 2. 10.1, 10.3, 11.3 3. + Resources | 1. **TCI project continue** |  | 1. Code styles and their importance 2. Comment in testing (JavaDoc) 3. Stress Tests 4. JMeter    1. Run test    2. Reports |
| 1. **6** | 1. review on week5 |  | 1. own selection of subjects needed for project. | 1. **TCI project continue** |  | 1. Maintainable tests 2. Simple cases to break 3. Empiric unit testing 4. Automated tests |
| 1. **7** | 1. review |  | 1. work on project | 1. **TCI project continue** |  | 1. **TCI project showcase** |
| 1. **8&9** |  |  | 1. - |  |  | 1. **Hand in TCI project (Monday w8)** 2. **individual assessment (w8/9)** |

1. \* = optional assignment, strongly recommended.

# The final project assessment

1. **DEADLINE for this project is MONDAY in WEEK8 at 8.00h!**
2. Deadlines are strict. It is not possible to hand it in late!
3. The assessment is run on the final project developed by the students only.
4. There are general requirements for the course, shown below, and specific ones for the assignment (see the assignment in week2). The general requirements also apply to the group assignment.

### Course Requirements

|  |  |  |
| --- | --- | --- |
|  |  | MoSCoW |
|  | Learning goal | M:6,S:7,C:8,W:9+ |
| general |  |  |
|  | local TDD and automated testing | M |
|  | local TDD, shared automated testing | S |
|  | local TDD, shared CI | C |
|  | extra's | W |
|  |  |  |
| CI & build | setup local CI environment using gradle only | M |
|  | setup local CI environment using gradle only | S |
|  | use IDE for Sw development | M |
|  |  |  |
|  | setup local CI environment using gradle and Git | S |
|  | setup local CI environment using gradle and Git | S |
|  | deliver description & config of local setup | M |
|  | deliver description & config of shared setup | S |
|  | building pipelines for CI tasks | S |
|  | gradle dependency management for project | M |
|  | gradle build,test from commandline | M |
|  | gradle continuous build,test commandline | S |
|  | gradle deploy to local environment | C |
|  | gradle deploy to other platforms | W |
|  | setup local CI environment using gradle and Git | S |
|  | setup local CI environment using gradle and Git | W |
|  |  |  |
| OTAP | use DEV environment | M |
|  | use TST environment | S |
|  | use ACC environment | S |
|  | use multi user TST environment | S |
|  | multiple target environments | C |
|  | multiple target platforms | W |
|  |  |  |
|  | setup shared CI environment | S |
|  |  |  |
| TDD |  |  |
|  | define purpose of classes and methods | M |
|  | define behaviour of classes and methods | M |
|  | create tests based on behaviour | M |
|  | use tests to create code implementing behaviour | M |
|  | explain TDD process versus own way of working | M |
|  |  |  |
| junit | use terminology used for testing | S |
|  | make use of unit tests testing state | M |
|  | make use of unit tests using mock(s) | M |
|  | use parameterized tests | M |
|  | use matchers in tests | S |
|  | create custom matchers in tests | C |
|  | use rules in tests | C |
|  | create custom rules | W |
|  | use code coverage tools | M |
|  | mutation testing | W |
|  | performance/stress testing | W |
|  | use exception testing | M |
|  | use tools for specific tests | C |
|  | use assertion messages | S |
|  | use asynchonous tests | C |
|  | use multithreaded testing | W |
|  | use time dependent tests | C |
|  | use collections or file tests | W |
|  |  |  |
| group work | see group assignment requirements in week2 |  |
|  |  |  |