ST COURSEWORK DESCRIPTION

February 25, 2021

1 SOFTWARE TESTING: PRACTICAL

This page describes the practical for the Informatics Software Testing course. It will be marked out of 100 points, and is worth 45% of the assessment of the course.

2 DEADLINE

The coursework comprises 3 tasks with the following issue dates and deadlines:

- Issued: 26th February
- **Deadline : 26th March, 4pm GMT**.

The policy for late submission has been modified this year because of the pandemic, rather than following the UG3 course guide (`http://www.inf.ed.ac.uk/teaching/years/ug3/CourseGuide/coursework.html`)

Coursework will be scrutinised for plagiarism and academic misconduct. Information on academic misconduct and advice for good scholarly conduct is available at `https://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`.

3 TOOLS

You can choose to use the Eclipse IDE, another IDE such as IntelliJ IDEA, or just to use JUnit and other appropriate tools of your choice standalone. I have no strong preference – many people find the tools available in Eclipse useful (if you haven't used Eclipse before maybe now is the time to give it a try). You can discuss your choice and problems you come across with the tools (without sharing specifics of your coursework solutions) on Piazza, but note that tool-wrangling is part of what this coursework is intended to help you learn so it is ultimately **your responsibility** to get your chosen tools to work. Don't leave it to the last minute!

The rest of this handout is written mostly as though you were using Eclipse.

You will need some of the following:

1. JUnit 4. If necessary you can download JUnit from `https://junit.org/junit4/`. If you are using Eclipse it is probably already installed in the IDE. This article: `https://www.vogella.com/tutorials/JUnit4/article.html` is a reasonable introduction to using JUnit4 with Eclipse.

2. You will need some kind of coverage analysis tool:

   - In Eclipse you can use EclEmma: `https://www.eclemma.org/`. If you use Eclipse on DICE, or if you install the version of Eclipse that is current at the time of writing (2020-12) and select "for Java developers" when asked, this should already be installed, and is accessible as "coverage" from the toolbar or Run menu. If not, it's easy to install through Eclipse's built in software update mechanism. IntelliJ IDEA also has a built-in coverage module.
   - For stand-alone coverage you should consider something like Cobertura: `http://cobertura.github.io/cobertura/`
   - A review of other OpenSource code coverage tools for Java is available at `https://java-source.net/open-source/code-coverage`.

Most of the tasks have an associated tutorial which will help you prepare for it. Please prepare in advance for the tutorial to get the most out of it.

4 SETTING UP

4.1 Preparation

If you don't have Eclipse installed and want to use it, you should download it and install it. You can find Eclipse at `https://eclipse.org/users/`. You want "for Java developers" when asked. This will come with JUnit and EclEmma pre-installed. Eclipse's help menu leads to a lot of information, some of which is outdated: you may wish also to consult the links given above.

5 TASKS

5.1 TASK 1: FUNCTIONAL TESTING (25 POINTS)

In this task you will implement JUnit tests using the specification provided in the Github repository (`https://github.com/SoftwareTestingEdinburgh/STCOURSEWORK2021`). The repository also provides the implementation as a JAR file, `ST_COURSEWORK.jar`, so you can execute your JUnit tests and observe test results. The specification is described in detail, with helpful examples where necessary, in the `Specifications.pdf` file.

Functional testing is a black box testing technique, so use the specification file to derive tests and **NOT** the source code. The JAR file can be used to execute the tests derived from the specification. We have also provided a sample JUnit test case, `CommandLineParserTest.java` file, to illustrate a typical test case for the implementation in `ST_COURSEWORK.jar`. All the files referred to above can be found at the Github repository (`https://github.com/SoftwareTestingEdinburgh/STCOURSEWORK2021`).

For this task, our `ST_COURSEWORK.jar` file contains 13 different bugs. The estimated difficulty of finding these bugs ranges from easy (5 bugs), to medium (4 bugs), to hard (4 bugs). Easy bugs will rewarded as 1 point, Medium bugs will be rewarded as 2 points, and Hard bugs will be rewarded as 3 points. If you are able to find all of them, you will get full points(25) from this task. "Finding" a bug means writing a test which should pass according to the specification, but fails on the provided implementation.

Deliverables:

- A file Task1_Functional.java that contains your JUnit tests.

SUBMISSION: Submit on Learn, just as you have for tutorials.

6 TASK 2: COVERAGE ANALYSIS (30 POINTS)

6.1 TASK2.1: Analyzing Code Coverage (10 POINTS)

The goal of this task is to measure and analyze the branch coverage of the STCOURSEWORK2021 code achieved by executing the test cases developed in Task 1.

Deliverables

- A file `task2-1.jpg` containing a screenshot of the branch coverage report as shown by the coverage measurement tool. Please make sure that the total branch coverage is clearly visible in the screenshot.

SUBMISSION: On Learn.

6.2 TASK2.2: Generate Automated Test Cases with EvoSuite (15 Points)

For this task, you will use EvoSuite to generate test cases for the project. EvoSuite is a tool that automatically generates test cases with assertions for classes written in Java code. To achieve this, EvoSuite applies a novel hybrid approach that generates and optimizes whole test suites towards satisfying a coverage criterion. You can find detailed information at `https://www.evosuite.org/`.

For a modern Java version, you will need to use the latest version, 1.1.0, of evosuite. At the time of writing, this is not available as a maven plugin, so you will need to follow the instructions for using evosuite from the commandline. Then, if you are using a coverage tool in an IDE, you will need to import the generated tests into the IDE.

Deliverables:

- A folder `Task2-2`. This folder should contain
    1. A file `task2-2.jpg` which shows the branch coverage report achieved by EvoSuite test cases. Please make sure that the total branch coverage is clearly visible in the screenshot.
    2. All of the generated test case files that were generated by EvoSuite.

SUBMISSION: On Learn.

6.3 TASK2.3: Advantages and Disadvantages of automated test generation Tool (5 points)

In this task, you will state three advantages and disadvantages of using automated test generation tool. Focus on questions such as:

- Why do we use automated test generation in the real world?
- How effective is it to use test generation tools for big projects?

Deliverables: A *plain text* file `Task2_3.txt` with a brief reflection, organized as a list of bullet points, of advantages and disadvantages of using automated test generation tools such as EvoSuite.

SUBMISSION: On Learn.

7 Task 3: Adding Functionality with Test-Driven Development (45 points)

A TDD approach is typically interpreted as "A programmer taking a TDD approach refuses to write a new function until there is first a **test that fails** because that function is not present." TDD makes the programmer think through requirements or design before they write functional code. Once the test is in place the programmer proceeds to complete the implementation and checks if the test suite now passes. Please keep in mind that your new code may break several existing tests as well as the new one. The code and tests may need to be refactored till the test suite passes and the specification is fully implemented. In this task, you will need to follow the TDD approach to implement and support a new additional specification in the existing implementation. The new additional specification is described in the last two pages of the specification file, available in Github repository (`https://github.com/SoftwareTestingEdinburgh/STCOURSEWORK2021`).

Deliverables: This task will involve a 2 part submission.

1. **Part 1**: Tests

   Submit **only your new JUnit tests** in a file named `Task3_TDD_1.java`, for the new additional specification (last 2 pages of the specification document). Please check to make sure all of these new tests fail on the existing implementation available in the `src` folder.

2. **Part 2**: Implementation + Tests

   This part requires that you add source code to `Parser.java` in the `src` folder to support the new specification. Check whether all the tests you developed in Part 1 pass for your modified implementation. If they don't, modify the implementation and/or tests so the entire test suite passes and the new specification is implemented correctly. Name your modified collection of new tests `Task3_TDD_2.java` (even if it is, in fact, identical to the file you submitted as `Task3_TDD_1.java` in Part 1). Submit both the modified implementation of `Parser.java` and the test suite `Task3_TDD_2.java`.

PS: For Task 3, your require implementation need only modify the file `Parser.java`. Please do not touch other java files.

SUBMISSION: On Learn.