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 25% of the assessment of the course. This practical will be undertaken individually or in groups of 2, and will be assessed on the basis of the group submission.

DEADLINE

The coursework is comprised of 3 tasks with the following issue dates and deadlines,

Issued: 25th February.

Deadline : 29th March, 6pm.

The penalty for late submission follows the uniform policy described in the UG3 course guide. **Coursework will be scrutinised for plagiarism and academic misconduct. Information on academic misconduct and advice for good scholarly conduct is available here .**

BACKGROUND

In this practical you will consider the specification and Java program available at the following github repository.

TOOLS

You can choose either to use the Eclipse IDE or just to use JUnit and other tools 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 will need some of the following:

1. If necessary you can download JUnit from here. If you are using Eclipse it is probably already installed in the IDE. This article is a reasonable introduction to using JUnit with Eclipse, but bear in mind its age: in particular it's focused on JUnit 3. Here's a good introduction to JUnit 4(free registration required).
2. You will need some kind of coverage analysis tool:
    - In Eclipse you can use EclEmma. This should already be installed on DICE machines within Eclipse. If not, it's easy to install through Eclipse's built in software update mechanism.
    - For stand-alone coverage you should consider something like Cobertura.
    - A review of other OpenSource code coverage tools for Java is available here

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.

SETTING UP

Preparation: If you don't have Eclipse installed and want to use it, you should download it and install it. You can find Eclipse here. Once you have installed Eclipse, you should look at the tutorial. Do enough of the "getting started" tutorial that you have JUnit as a project in Eclipse. You should also install eclemma if you don't have it and intend to use it. You can delay this since it is not essential for the first task.

You should spend some time looking at the JUnit project in Eclipse and become familiar with its structure.

Please use JUnit 4 as your testing environment.

TASKS

TASK 1: FUNCTIONAL TESTING (25 MARKS)

In this task you will implement JUnit tests using the specification provided in the Github repository. 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 under the `jar` directory 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.

In giving a grade for this part of the practical I will take into account the performance of your test set on a collection of variants/mutants of the specification.

Deliverables:

1. A file `Task1_Functional.java` that contains your JUnit tests.

TASK 1 SUBMISSION:

To submit your work you should designate **one member of the group** as a submitter for the group. The submitter will gather together the files you wish to submit for Task 1 and execute this command.

```
submit st cw1 Task1_Functional.java
```

TASK 2: COVERAGE ANALYSIS (25 MARKS)

Using some appropriate coverage measurement tool (such as EclEmma mentioned in "Tools" and "Setting Up"), assess the following:

1. Take the JUnit tests you developed in "Task 1 Functional Testing", and measure the **branch coverage** achieved. Run the JUnit tests on the source code of the implementation, which is available in the folder "SourceCode/src/st" in the

Github repository. Submit a screenshot showing the coverage achieved by tests developed in "Part 1 Functional Testing", as reported by a coverage measurement tool like EclEmma.

2. Attempt to improve the branch coverage achieved over the source code to "maximum possible" by adding more tests. You can look at the source code and its structure to guide the development of additional tests. Please note that "maximum possible" branch coverage may be less than 100% since there may be parts of the code that are unreachable in the provided implementation. Re-assess branch coverage achieved with these additional tests along with tests from Part 1. Please submit **all the JUnit tests** used to achieve maximum coverage, and a screenshot showing the improved branch coverage as reported by a coverage measurement tool like EclEmma.

Deliverables:

1. Screenshots showing the different levels of coverage you achieved (coverage_1.jpg and coverage_2.jpg).
2. A file named `Task2_Coverage.java` containing **all the JUnit tests** used to achieve maximum coverage.

TASK 2 SUBMISSION:

To submit your work you should designate **one member of the group** as a submitter for the group. The submitter will gather together the files you wish to submit for Task 2 and execute this command. The dots at the end of the command signify any other relevant files:

```
submit st cw1 Task2_Coverage.java coverage_1.jpg coverage_2.jpg ...
```

TASK 3: TEST-DRIVEN DEVELOPMENT (TDD) - (TOTAL 50 MARKS, GROUP ACTIVITY)

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 he/she writes 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 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.

Deliverables:

This task will involve a 2 part submission,

1. **Part 1 :** Tests
   Submission for this part will be **only new JUnit tests** , name

it `Task3_TDD_1.java`, for the new additional specification (last 2 pages of the specification document). Please check to make sure tests developed for the new specification fail on the existing implementation available in the "st" folder.

2. **Part 2 :** Implementation + Tests
   This part requires that you add source code to `Parser.java` in the "src/st" folder to support the new specification. Check if all the tests developed in Part 1 of TDD pass for the 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. Submit both the modified implementation of `Parser.java` and the test suite from part 1, `Task3_TDD_1.java`, including any revisions or modifications as `Task3_TDD_2.java`
   Execute the submit command shown below.

SUBMISSION:

To submit your work you should designate **one member of the group** as a submitter for the group. The submitter will gather together the files you wish to submit for the 3 tasks, and execute this command. The dots at the end of the command signify any other relevant files:

```
submit st cw1 Task3_TDD_1.java Parser.java Task3_TDD_2.java ...
```