COMP1003 CW2 – JUnit Testing

20-3-2024 – Max L. Wilson

Starting Premise: It turns out that, since the last meeting, mediocre developer Lax W. Milson has been just coding away with no plan. When Yu Rong - the Quality Assurance Manager - saw Lax's code in a Fagan Inspection, she realised the code that he produced was terrible in more ways than one – almost as if it was built badly on purpose. In keeping with her name, she told Lax that he wasn't doing it right, and assigned Mayhem Grutton as the new Dev Team Leader to sort it out. Mayhem has produced some detailed initial Class Descriptions, and decides the team should do test driven development, and so brings in the head of testing: Mary Bugjahan. Mary wants your team to begin devising tests to fix the bad code. Once you've made that plan, and expects you to start fixing the code following proper Test Driven Development expectations (although ideally no code should have been written before the tests, TDD can at least help rectify the situation). As with the whole company, Yu Rong expects that you will follow good development practices, in particular: code reviews (where any submitted code is reviewed by at least one other person, before being merged), and detailed commenting (e.g. author of each test or code change, change history, cross referencing with documentation, etc). Mayhem expects proper use of version control software (e.g. branches from issues, using labels, milestones, etc).

First Lab: Take the new draft Class Descriptions document and prepare a rigorous set of tests for how the code *should* work. After the first lab, you'll be using these tests to improve the code. Therefore, you should:

» Produce a table of planned tests in markdown, using the markdown template provided.
» and add rows, columns, and more tables as needed.

It would be recommended to spend the first lab discussing how to do this work effectively, and to begin the test plans with good strategy. You should also use some time to agree on milestones, how code reviews will be organised in the team, and how you will use e.g. labels to do so.

Middle of first lab: You will receive Lax's dodgy code, and you may wish to use this to identify further tests. You will also receive guidance for setting up your repository so that the java files synchronise, but not the java project metadata and compiled code etc. You may wish to begin to divide up work and plan how you will do the coursework well together.

By the end of the first lab:
You should have created a markdown report (in the docs folder) of your test plan for each class.

After first lab: The main task, for which you will be primarily assessed, is the development of good tests in JUnit. You have three classes (the dodgy code built by Lax) to create tests for. You should try to build tests according to your test plan, but it is normal to realise more tests could be useful, or that some are not needed, etc, as you start writing tests and debugging.

Using a) Issues, b) comments in the code, and c) detail in the markdown test report:

- You should document whether/when tests pass/fail.
- If failed, you should document the cause, what was fixed, and when they then passed, and then perhaps create a bug issue for it to be fixed.
- Further, in e.g. code comments, you should note who in your team authored each test, and when etc.

Main Task Stages:

Unit tests (approx. 1 Week): You should start by creating JUnit tests for the two simpler child classes (used by the main app).
- you may want to finish this before the mid-way lab, so you have a week to a) fix up the code, and b) do the harder integration testing.

Integration tests (approx. 1 week): After finishing the first simpler classes, you should move onto Integration Testing, by building JUnit tests for the Main Class, which uses the other two classes**.**

Please Turn Over

Secondary Task: You may also start improving and fixing the bad code produced by Lax. The well-designed tests should reveal all the bugs that seem to exist.
 - As well as creating bug issues (and branches from them), you should use comments in your code to identify what changes were made, and by whom.

Tertiary Task: You must submit three further peer assessments so that we know who contributed properly to the project. These should focus on the whole of that week.

- After 1st dev week (before easter)
- After 2nd dev week (after easter
- After submission.

Submission to gitlabs:

1. Source code in src and tests folders, containing: java code and junit tests.
2. A markdown report (in the docs folder – an evolution of your test report), with pass/fail log, identifying the date that tests passed, and using cross-referencing in some form with the code and gitlab issues etc.

You do not need to (should not) build continuous integration pipelines etc.

Deadline: The deadline for this work is May 9th, 6pm.

- You should tag your repo with `CW2.0` by this time, only when the team is ready to 'submit'.

Asynchronous Group Work & Timing: This is group work, and you are expected to work on this in your own time between now and the deadline (although there is no need to work during the easter break). You must coordinate yourselves in your own time. The 4x2hr lab sessions are not enough, alone, to achieve a good grade.

Additional Help Sheet: There will also be an optional lab sheet released just before easter to help with the integration testing.

CW2 Marking Criteria: In this work, you are being assessed primarily on your ability to write and use Unit tests, and then by how you do this as a team of remote asynchronous collaborators.

1) The majority of marks are associated with the quality of your JUnit tests, including **coverage and rigour** of tests, and appropriate use of a range of different JUnit features, beyond just the basic @Test with AssertEquals(). Top Marks are for achieving the hardest integration testing.
2) Less than half the marks are associated with **Good SE Practices**
    a. Evidence of a Code Review process being used (and e.g. how the code appears as the work of a single team, rather than code written in different styles).
    b. How teams use version control, issues, branches, labels, milestones, etc.
    c. If (and how) teams fix bugs, and document code changes.
3) The remaining marks are associated with the detail captured in your **Test Report**, about when and why tests fail, and what changed to make them pass.

Estimating individual contributions: We will be using evidence from the following information to decide whether people contributed effectively:

- Activity in git repositories (commits, issues, responses, etc)
- Peer assessments
- Author-attributing comments in code and tests

It is your responsibility to make sure your contributions to this coursework are visible. **All people should be involved in all aspects: writing code/tests, code review, and documentation.**