Module Code
COM00122M

BSc, BEng and MEng Degrees Examination 2018—9

DEPARTMENT OF COMPUTER SCIENCE

Software Testing (SOTE)

Open Individual Assessment

Issued: 13th March 2019, 12:00 noon
Submission due: 17th April 2019, 12:00 noon
Feedback and Marks due: 8th May 2019, 12:00

All students should submit their answers through the electronic submission system: http://www.cs.york.ac.uk/student/assessment/submit/ by 12:00 noon, 17th April 2019. An assessment that has been submitted after this deadline will be marked initially as if it had been handed in on time, but the Board of Examiners will normally apply a lateness penalty.

Your attention is drawn to the section about Academic Misconduct in your Departmental Handbook: https://www.cs.york.ac.uk/student/handbook/.

Any queries on this assessment should be addressed by email to Dr Steven Wright (steven.wright@york.ac.uk). Answers that apply to all students will be posted on the VLE.

Rubric:

Carry out the whole task as described in the following pages. *Your report must not exceed 10 sides of A4*, with a minimum 11pt font, minimum 120% line spacing (what Word calls “Multiple 1.08”), and minimum 2cm margins on all sides. This does not include any covering page, table of contents or reference list. **Excess pages will not be marked.**

References must be listed at the end of the document and do not count towards page limits.

Your exam number should be on the front cover of your assessment. You should not be otherwise identified anywhere on your submission.

Testing a spell-checker library

The assessment consists of testing JOrtho, a spelling-checking library coded in Java. JOrtho version 1.0 is available at http://sourceforge.net/projects/jortho/.

It may be helpful to imagine that you have been commissioned by someone in the same organisation, as a testing expert, to tell them about the software quality of the software under test. They have given it to some potential users (application programmers who would like to integrate it into the application they develop) and are reasonably confident that it provides the general features they require. They would like to know if they can really trust the system to be dependable, before they commit themselves to using it several projects. They therefore want you to provide a thorough assessment of its freedom from defects.

You can assume that JOrtho should function similarly to spell-checkers in other applications, such as available commercial word processors. However, when deciding what to test and what the correct behaviour should be, you should also consider likely user assumptions, the behaviour of similar libraries, and the uses implied by JOrtho's demo programs and documentation. See lessons 31-35 in the Kaner et al. book[1] for some advice on deriving requirements as part of the testing process.

You will notice that the Subversion repository for JOrtho contains some existing test code in JUnit 3 form[2]. Your testing should not duplicate these test cases – you will not receive any marks for tests that do.

Scope of testing

The whole of JOrtho is in scope. Although it is not a large library, you will still find that you have too much to test thoroughly in the time allotted (or write up inside the page limit). You should prioritise what you test, and justify that in your test plan.

Testing Report

Your assessment submission is a report consisting of the following sections.

Section A — Test Plan [25 marks]

A test plan. It should detail the method(s) used to build the test cases and the software tools to use to achieve these.

1. Clearly define what constitutes "the software under test", and list the features that you will test and you will not test.
2. Explain how you have determined the expected behaviour of the software (in the absence of an exhaustive and explicit requirements specification).

[1] Kaner, Bach and Pettichord, Lessons Learned in Software Testing: A Context Driven Approach. John Wiley & Sons, 2002.
[2] http://sourceforge.net/p/jortho/code/HEAD/tree/trunk/JOrtho/test/com

3. Explain the overall strategy you used for creating test cases, and for selecting the specific test cases that you present in section B.
4. Define acceptability criteria for the software – what (testable) properties does it need to have in order for it to be of acceptable quality for its intended purpose?

Section B — Test Case Specifications [30 marks]

Test case specifications for 8 fully specified test cases. The test cases must be complementary: they must each make different assumptions and test different specific features of the library.

1. At minimum, each test case should describe the stimulus applied to the software (which might be a sequence of API calls, a sequence of user actions that are taken, or something else) and the expected i.e. correct behaviour.
   - NB for some tests it may be appropriate to define constraints (e.g. "no files will be changed") instead of/as well as positive statements of behaviour (e.g. "the user will be returned to the main menu screen").
   - Where the reason why the expected/correct behaviour is indeed expected and correct is not obvious to a capable programmer with some domain knowledge, explain briefly why it is so.
2. Each test case should also state the *purpose* of the test within the test set. A good way to do this may be to state the question that the test case asks about the software. If we cannot understand what the purpose of a given test case is, we cannot give you much credit for it.
3. "One test case" should test one thing – one feature, one unusual input, or one user task. For example, if you have a function that takes an integer as an input, testing it with Min/Max/+1/0/-1 should be five test cases.

   One can note that those are, however, five *very low-level* (unit test) test cases, which are unlikely to give you the most testing power in a fixed number of tests, and hence unlikely to give you maximum marks.
4. You should aim to provide a diverse range of test cases, and also to provide your best test cases (including any that find interesting bugs). There is an inevitable tension between these two objectives, which you will have to decide how to resolve.
5. For full marks in this section, tests should be conducted at a range of testing levels, including unit, integration and system level. *Hint:* explicitly label each of your tests with what level it's working at.

6. You may include short fragments of code within your test case specifications, but not larger ones. In particular, do not include whole JUnit tests. In essence, you need to provide enough information for a smart programmer to recreate your test code given some effort. E.g. if a test case involves a specific sequences of method calls, that sequence of method calls needs to be clear from the test case description.

Section C — Test Results [15 marks]

The test results for each of the tests that were specified in item (B).

Here, you should document the results that occurred when the test cases are run. You should provide explicit indication of whether each test passed or failed, and in the latter case state what happened instead.

Section D — Test Summary Report [30 marks]

A test summary report that will contain at least:

1. a summary of the testing that you performed
2. a summary of the results you observed, including a classification of the faults found (by an appropriate classification scheme of your choosing)
3. an overall evaluation of the thoroughness and quality of the testing you have performed
    a. This should be in terms of what might be possible given substantial resources, not in terms of what is possible in a project with this time allocation and maximum report length.
    b. Describe the branch coverage *and* condition coverage, or the mutation score according to a reasonable mutation testing approach, that your tests achieved of the Java code (not of any other artefact). Briefly explain how you did it; make sure you state what tools you used and (where applicable) what mutation operators you used.
4. an overall evaluation of the software tested (in terms of its freedom from faults)

Throughout, the summary report should only refer to the test cases that were specified in (B), not any other testing you may have performed.

Some General Advice

- You won't receive marks for testing that the marker merely *thinks* you probably did — marks will only be awarded for tests that are described explicitly and precisely.
- Yes, it is a little unrealistic that you are only allowed 10 pages and 8 tests, and that your possible coverage and comprehensiveness are limited by that. However, in the real world there are always resource limits — the ones you have here are merely artificial ones.
- Do not repeat anything that does not vary. For example, do not repeat the same boilerplate text in every test case description — if something is true for all test cases, say this once at the start of the section.
- Similarly, use blanket statements to make points that are true of multiple (but not all) tests case (e.g. "Test cases 1–5 assume that…")
- Appendices containing full JUnit code or detailed test output etc are not required and will not be read.
- In general, do not waste space on spurious information. Think about what the notional target reader needs to know in order to use your document, and include only that. At best, other information will waste some of the limited pages available to you.
- If you reference an external document (such as the documentation or website for the software you are testing), be sure to cite it appropriately, just as in any other submitted work.

END OF PAPER