UG Exam Performance Feedback
Third Year
2018/2019 Semester 1

COMP33711 Agile Software Engineering

Suzanne Embury
Christos Kotselidis

Comments See the attached report.

Feedback on Student Performance in COMP33711 Exam

January 2019

Suzanne M. Embury
Christos Kotselidis

Question 1

The students have shown a great understanding of the agile principles and their performance was excellent for the first question.

Regarding the second question of Part A, the students shown great imagination in proposing user stories and potential mitigation strategies in part c.

One general comment that extends to a vast majority of answers is the granularity of user stories compared to epic. Students tend to propose complicated user stories instead of really small ones that we could easily use also for epics.

Question 2

There were many very solid answers to this question, indicating that a significant subset of the candidates had understood the material well, and were able to apply the techniques to simple examples. Unfortunately, there were also a number of candidates who failed to answer large parts of the question, seeming to have very little understanding of what the terms BDD and TDD actually mean.

a) There were many excellent answers to this question, which was designed to test basic understanding of the core concepts behind specification-by-example. Marks were lost through making vague and imprecise statements ("examples help developers understand the system better") or through overly strong (and therefore incorrect) statements ("examples aren't ambiguous but natural language documents are", "examples can't be misunderstood by customers").

b) The overall quality of answers for this question was very high—higher than I'd been expecting given the very small amount of time we had to cover these concepts. Most candidates were able to diagnose the problems with the missing When step in the first scenario and the technical language used in the second. By contrast, only a handful of candidates correctly diagnosed the core problem with the third scenario, which was that it was written like a script, rather than as a declarative example. Despite this, many candidates recognised that the fix was to break the script into multiple scenarios that respected the G-W-T ordering.

Incorrect diagnoses were given by some candidates, however, with a significant number of candidates adding more problems into the scenarios than they removed by rewriting them. The most common errors/misconceptions are listed below.

A number of students complained (incorrectly) that multiple cases were being tested in some of the earlier examples. This would have been a very valid criticism if we were talking about unit tests, but different criteria apply to acceptance tests—especially when they are intended to be read by customers. Readability and elegance of the expression trumps diagnostic power in this case. It is also the case that many acceptance tests require fixtures that are quite slow to set up (adding data

to a database, for example, or creating large input data structures). In this case, we often want to reuse fixture for multiple cases, because this allows us to test more cases in the time available than if we had kept them strictly independent.

A small number of people were worried about processing the dates given in the scenarios, and wanted to convert them into formats that were simpler to parse. Once again, this is not correct for acceptance tests, which need to readable for the customer before they are easy for the developer to write glue code for. (Besides that, there are many libraries available that will parse any date in ISO format, including those given in the example scenarios. No additional programmer time is needed for this format of date than for the simpler ones candidates wanted to replace them with.)

Other candidates were worried about just specifying dates using days (saying “Monday” instead of “4th February 2019”). This is again a matter of readability. If it is clear to the customer what behaviour is meant when using only days, then that is how the scenario should be written. There was no hint in the question that customers had any problems with the date formats used.

Some candidates seemed to struggle with the When parts of the rewritten scenarios (even for the scenarios given in the question that had perfectly reasonable actions). These were replaced with phrases like “When the page is refreshed” or even in one case “When the dashboard is refreshed”. This was surprising as the question makes no mentioned of pages (or dashboards) being refreshed. Perhaps students who went down this route were borrowing too heavily from answers to last year’s revision question, which included a board with ticket prices that needed to be refreshed. Obviously, this year’s exam question was about an application that could be implemented through the web, and that therefore would involve page refreshes, too. But we wouldn’t expect users of the system to be interested in this low level technical detail, and we also wouldn’t want to embed assumptions about the implementation technology into the specification.

The final common misconception was a complaint that some of the Given steps described actions, when only a When step can describe an action. This was true in the case of the final scenario, where the poor author was desperately trying to make every step into an action in the “script” she was trying to express, but the same complaint was made about the other two scenarios. Given steps describe conditions on the state of the system; the state of the system is affected by actions being applied to it. Therefore, it is perfectly legitimate to have Given steps that ask whether the system state records whether an action has occurred or not.

c) i) There were a lot of really excellent answers to this question, selecting simple happy path tests that were completely in line with the small amount of specification given in the question.

The most common mistake in selecting the test was where candidates failed to read the question carefully and wrote a test for some part of the functionality not related to the notification behaviour requested. There were several solutions that gave unit tests for behaviour to do with adding and removing tags, for example.

Others failed to remember that any first test that asserts that the notification check method should fail will probably pass even before any code has been changed in the coding step, because most IDEs will produce stubs for Boolean methods that return ‘false’ as their result. Since we did not specify this in the question, however, no one was penalised for this.

The most common programming-by-wishful-thinking error was to omit information about the post or the possible recipient of the post (or, in some cases, both) in parameters of the method that checks whether notification should be performed or not. Another common error was to forget to connect the post to the user in some simple way (e.g. by setting the user as a recipient of the post).

A basic coding error that was made by some students was to set up local variables in an @Before set up method, and to assign values to them (e.g. Post post = new Post(“message”)) and then try

to use those values in the body of the @Test method—even though they were clearly defined as local variables in the @Before method.

Finally, a very small number of candidates gave step definition code for their answer to this part of the question. I can only assume that these candidates had memorised the answer to the sample exam paper and/or the revision questions and had just decided to hope that I would ask for exactly the same things again this year…

ii) Unsurprisingly, the most common error for this part of the question was to write production code that did not closely follow the test. That is, to write production code that did a lot more than the single line of code needed to make the test pass. Most students seem to have grasped the concept of "fake-it-till-you-make-it" in TDD, however, and were able to give the minimal example.

iii) There were some excellent answers to this question, demonstrating a firm grasp on the early stages of the TDD. Other candidates struggled, and wrote production code when they should have written test code, and step definition code when they should have written production code.

A surprising error, made by several students, was to give a test that was exactly the same (textually) as the unit test code given in answer to part i) with the exception that 'assertTrue' was replaced with 'assertFalse'. Since everything else about the test was the same, it was not clear how the candidates who wrote these test expected to make them both pass at the same time.

Many candidates removed the few sensible refactoring options by rushing to write separate fixture code in the test class in their answer to part i). This left no meaningful duplication between the two test cases, that could be refactored out once the duplication became visible. (I would have accepted an answer that no refactoring was possible in these cases, if accompanied by an explanation that the main refactoring chances were already implemented in the first test case.)