UG Exam Performance Feedback
Third Year
2017/2018 Semester 1

COMP33711 Agile Software Engineering

Suzanne Embury
Christos Kotselidis

Comments Please see the attached report.

Feedback on Student Performance in COMP33711 January 2018 Examination

Suzanne Embury and Christos Kotselidis

February 2018

Q1.

Students performed exceptionally well and showed a great understanding of the agile values with clear and elaborate justifications when asked.

Regarding the VideoNet use case, many students assumed that the new electronic system would run in parallel with the old telephone-based one. Although it was not explicitly mentioned, several user roles and stories given (e.g. telephone operator, etc.) were considered correct.

A minor comment for some students regarding the poker-planning of question 2 iv): the points awarded during poker planning do not concern the value of the story but the technical difficulty of the story. The value of the story has been factored in the selection process (a bit earlier). During user story estimation, we assess the technical difficulty of the selected user story.

Q2.

There were many excellent answers to this question, with an unusually high number of candidates getting first class marks for this question. Unfortunately, there were also a significant number of candidates who did very poorly, and who scored less than the basic pass level for the question. Given that attendance was very low for the workshops after reading week, which this question covered, this is perhaps not surprising.

a) This was supposed to be an easy starter question, but in fact turned out to be the part of the question that was most poorly answered. Most (but not all) candidates were able to name 3 agile practices that could be applied in the early stages, but fewer were able to justify why these practices would help in the specific case described in the question. To earn marks, the practices needed to be linked to at least one of the two problems mentioned: the client's lack of confidence in the agile approach and the specific client concern that the complex business rules in the domain would not be implemented correctly. Many candidates gave only vague justifications. For example, "User stories will help the team get the business rules right". Maybe so, but how? It's not enough to state that the practice will solve the problem. To earn marks, candidates needed to at least attempt to explain the mechanism by which the problem is solved.

A few answers revealed some misconceptions about some of the practices. The most common were:

- Claiming that retrospectives would help the team to check the correctness of the business rules. In retrospectives, we assess the process that the team is using, and not the details of the product being built. Any time discussing the correctness of specific business rules in their retrospective has missed the point of the practice.
- Claiming that daily standup meetings would help the team to check the correctness of the business rules. A daily standup is meant to be a very lightweight team progress monitoring practice, and detailed discussions of specific implementation issues should not be taking place. At most, a note can be made of an issue, and the affected team members should get together outside the daily standup to
- Claiming that TDD would help to reassure the client that the rules were correct. TDD is a technology facing practice, and not a business facing practice. It can help to reassure the team that the correct rules are implemented, but is not something that the client would normally see or get involved in (apart from when the practice raises discussions about the required behaviour).

b) Many students were able to give a good clear Gherkin scenario for this question, and a significant number earned full marks. The most common mistake, however, was to give a generic statement of the required behaviour instead of giving a specific example, without giving values for the quantities involved. For example, the following uses the broad Gherkin syntax but does not describe a specific example:

> Given a non-priority un-subsidized route
> When income > cost + threshold
> Then the route is profitable

The marking scheme allowed me to give some marks for these kinds of scenarios, but for full marks a true Gherkin scenario with actual values for income, cost and threshold (and a proper When step) was needed.

Unfortunately, there were others candidates who seemed not to know what a Gherkin scenario looks like, and were unable to write anything resembling a Gherkin scenario. This had a serious follow-on effect on the later sub-questions, where further marks were lost.

c) Most candidates who produced a sensible scenario for part b) were easily able to generate a set of further valid examples for this part of the question. Where candidates were giving scenarios in generic form, rather than as examples, marks were awarded when it was clear that a number of distinct cases were covered by the scenarios, and that the business rules described in the question were being followed.

d) There were a number of excellent answers to this question, showing that candidates had grasped the principles of acceptance test automation (and programming by wishful thinking) well. The most common causes of lost marks were:

- Being sloppy in writing the regular expressions for glue code methods, so that they did not in fact match any of the steps in the selected scenario. I did not penalise small errors of one or two characters, but in some cases the text of the regex was lacking several words from the step definition (or vice versa). The whole point of the regexps is that they should match the step text *exactly*.
- Putting groups in the regular expressions for glue methods when no values were given in the steps of the scenario being implemented. Candidates who failed to include values in their selected scenario could still earn full marks for part d) but only if they hard-coded suitable values into the glue code, to make up for their absence from the scenarios. Almost no one did this.
- A disappointing number of candidates included production logic in their glue code, typically by including code to calculate whether a route was profitable or not in the body of the @When or @Then glue methods.

Most of the domain code designs produced through programming-by-wishful-thinking were okay. The most common mistakes in this part of the question was to fail to link the route to the dashboard. Some students also appeared to struggle with writing the assertion to check what the dashboard would display (since we were using middle-out testing and therefore did not have direct access to any client performing this display). The designs that were produced were not perhaps the most elegant, but I did not penalise students for this, provided the basic check required was being made.

A surprising number of candidates failed to indicate in their answer which scenario they were implementing, which made marking the question much harder. I was forced to guess which scenario was intended; it was not always obvious, when the regular expressions were not written correctly.