Feedback on Student Performance in COMP33711 Examination January 2020

Question 1

a) Some students misunderstood this question and compared two other concepts from the course (most frequently Agile vs BUF/waterfall, TDD and BDD, or lean and adaptive development).

A number of students incorrectly focused on the medium of contract expression (i.e. verbal vs. written) or provided vague descriptions that could not be mapped to concepts from the examinable reading. Marks were given for clearly expressed unambiguous descriptions regardless of name used for that contract type; however, the term "flexible" was only allocated marks if the description mapped to a time and materials contract.

In general students who were able to describe two approaches (part i) were also able to map these to agile practices and/or principles (part ii).

b) Almost all students successfully described a system that would be useful and deliver value to the customer. However, a good proportion of the described systems weren't particularly minimal, and many were ambiguous in one or more implementation details (e.g. "the publishers will be updated with the new subscribers".... how?).

c) Students who did well on this question provided three stories with the following properties:

- Precise user whose role related to the story provided. Roles such as "Customer", "User", "Publisher" did not score marks for user role. Roles such as "Subscriber" or "Reader" (with no further detail) scored only if the story was clearly only relevant to an existing subscriber (i.e. the term wasn't simply being used as a catch all for site visitors).
- Distinct user roles in each story.
- Genuine user (i.e. non-development team) roles.
- Stories whose "so that" clause would have evident business value or behaviour change.
- Stories with a sensible suggestion of desired functionality.
- One story significantly larger than the other two.
- Epic story not just a restatement of problem.

Many students scored well on this question. However, of the above, user roles and epics were the most consistent problem areas.

d) This question required application of knowledge to an idea not covered in lectures and most students made a good attempt to do this.

Part i was successfully completed by most. A small number of students lost marks here, usually as a result of either:

- Listing questions that could not be answered by the customer (e.g. implementation details).
- Listing questions that could not be mapped to any of the written user stories.

Part ii also received good answers from the majority. A few more students lost marks than for i, usually as a result of failing to describe changes that would be made to the user story (e.g. simply listing the customer responses), or for saying that the story would change without saying what the change would be (e.g. “I would split this into smaller stories”... what would the resulting stories be?).

Part iii was more challenging than the two earlier parts. A small number of students misunderstood this question as requiring them to evaluate their provided user story using the INVEST criteria. Some students described the requirements in general terms rather than providing acceptance criteria. Since the question was marked generously, really fine-grained clearly specified requirements could receive some (but not all) available marks.

Question 2

Performance on this question was mixed. Some candidates were able to give excellent answers, showing a good understanding of how to write effective Gherkin scenarios and how to automate them through sensible and clear glue code. However, these were in the minority. A significant proportion of the answers suggested that candidates had only a superficial awareness of basic BDD concepts and how this testing practice dovetails with other agile practices. Candidates struggled most of all with writing glue code for their scenarios, with some candidates unable to write anything even resembling step definition code.

a) There was a wide range of answers to this question. Many candidates spotted the obvious opportunity here to both gather and test requirements through shared creation of a paper prototype followed by one or more Wizard of Oz testing sessions. For full marks, the answer had to make clear that the prototypes were working systems built on paper. Answers talking about wireframes and “whiteboard mockups” were given fewer marks, since it is not clear how these kinds of mock-up can be used to test requirements, as mentioned in the question, as well as to elicit them.

However, there was widespread confusion about what it is reasonable to ask an ordinary member of the public to contribute in such a context. Many answers suggested that the volunteers should be asked to state the requirements for the system, as though they would magically know what the charity wanted. It was surprisingly common to refer to these volunteers as “customers”, when the charity itself is the entity that is paying for the software to be delivered. Some candidates suggested that these volunteers could set the priorities for the work to be delivered, again implying that they have a crystal ball telling them what the charity itself would want to have.

Others suggested that the volunteers be asked to write Gherkin examples and acceptance tests and even unit tests (for TDD), which is asking a lot of untrained random people in a single day. Some candidates wanted to spend part of this day giving the volunteers an introduction to agile methods, though what use this would b when they were only participating for that one day was not clear. I allowed the suggestion that the volunteers

be asked to provide user stories, but only when it was the dev team who was doing the work of translating the discussion into stories, and not the volunteers themselves.

b) Many candidates were able to identify the problems in this scenario, though very few people were able to convincingly explain why the problem was considered bad practice.

Common mistakes made by candidates involved:

- Stating that Gherkin scenarios can have only one “Given” step.
- Stating that Gherkin scenarios may have more than one “Given” step but all but the first must be labelled with “And”.
- Stating that “And” steps are only allowed after a “Given” step.
- Stating that line 3 of the scenario was unnecessary (even though it is the only place in the scenario steps where the type of petition is specified).
- Stating that line 3 was unnecessary because the scenario title states the type of petition (when, of course, the scenario title is not part of the executable elements of the scenario and so doesn’t count).
- Stating that Given steps shouldn’t contain actions (which is strictly true but they can describe the requirement that an action has taken place).
- Stating that the presence of the second When indicates that there are two scenarios combined into one here: one relating to signing the petition and the other relating to checking the petition (when it was far from clear what “checking” a petition means – the ambiguity of that was one of the problems with the original scenario).
- Stating that the supporter’s name does not need to be given, because it is redundant/irrelevant (even though we need the name to be able to check whether the right values have been added to the petition after the supporter has signed it).
- Stating that it was a mistake to include specific values like the name of the supporter in the scenario, on the grounds that we need to write generic rather than specific scenarios (when, of course, the whole point of this specification-by-example approach is that we should include these specific details and should avoid trying to express generic requirements).

Many of the answers to this question suggested strongly that candidates had not really understood how Gherkin and Cucumber work, having merely skimmed through the slides for the course unit rather than actually attempting to write any Gherkin or glue code for themselves.

c) There were many adequate solutions to this question, though very few people managed to give scenarios that really captured the core behaviour. This is not surprising, since writing Gherkin is a difficult skill that needs much practice and experience to acquire. The marking scheme made allowances for this fact.

Most lost marks came from copying across the problems in the original version through to the rewritten version. For example, many candidates failed to notice the problem with technical language in the first version (“view web page”, “click button”) and allowed these unhelpful wordings to remain in the rewritten version. Others gave a scenario that was even less like a true example than the original, having no name or identifying values given at all.

A number of scenarios given ended abruptly, having collected some information about the supporter but not actually doing anything with it. This was common in the solutions given by people who tried to split the scenario into two separate scenarios, neither of which fully expressed the behaviour of signing a normal petition. (How can you check whether the action of signing a petition has been carried out successfully without looking at the list of signatories to the petition?)

Finally, a number of candidates wrote scripts that assumed a very specific GUI design and wrote steps describing how the supporter interacts with that GUI (e.g., "the supporter enters their name"). This leads to low level scenarios and tricky-to-write glue code. Scenarios should really be written in domain terms, not in terms of specific implementation choices.

d) Most candidates managed to provide two reasonable additional scenarios, though only a very few candidates expressed them clearly with all necessary details provided.

A number of candidates failed to reuse the step texts from their answer to part c), and instead came up with completely different wording for the same ideas for these scenarios.

Other common errors here were not writing end-to-end scenarios (e.g. scenarios that end with the user entering their post code, but then doing nothing with that information), and writing scenarios for other features when the question asks specifically for additional scenarios for the "sign petition" feature.

e) Answers to this question were mixed, with some candidates writing excellent and complete glue code and others seemingly unable to write anything even resembling glue code. This was the only part of question 2 where significant numbers of candidates were awarded 0.

The most common causes of lost marks in writing glue code were:

- Writing step annotations that don't actually match the wording of the steps in the scenario.
- Writing "..." in the step annotations instead of writing the step text out in full.
- Failing to use capture groups to pull out key values from the step text.
- Pulling out values from the step text using capture groups, but not actually doing anything with the values in the step definition method.
- Declaring parameters for the step definition method, even when the step annotation text does not contain any capture groups.
- Putting assertions in the Given step definitions, to check that the state is as we want it to be, but not actually taking any action to cause that desired state to come about.
- Failing to put any assertions in the Then step definition.

Marks lost in terms of programming-by-wishful-thinking/software design:

- Using non-domain terms for the classes (e.g. a class called "System" giving nonsense code like "system.addPerson(name)").
- Writing the step definitions at the outer level of the code (GUI) and not in terms of service level API classes (e.g, incorrect code such as "page = new WebPage(); page.clickButton()"). A really handy way to know if you are making this mistake is to

check whether you are writing classes/methods that are based on domain terms or if they are based on implementation concepts. The former is correct, the latter is usually an error.

A number of candidates seem to be assuming that no other scenarios would be implemented using their glue code, and therefore set about hard-coding lots of parts of the fixture that should really have been controlled by values embedded in the step text. For example, it was common for students to hard code the name of a Supporter in the field declaration. This means we can never write scenarios (for any feature in the system) that requires a supporter with a different name, or requires two supporters with different names.

I did not remove marks for this, since the question asked for glue code for only one scenario. But, for future reference, it is important to write the step definition methods as generally as possible, so that the step can be used in many different scenarios, possibly for many different features.