Comments

Please see the attached report.
Feedback on COMP33711 Agile Software Engineering January 2016 Exam

Q1.

a)

Although there were some excellent answers to this question, there were many very poor ones. The major cause of lost marks was failing to provide all the information requested in the question (i.e., poor exam technique).

The question asked for three things:

1. An explanation of why the concept reduced waste.
2. A named agile practice that reduces waste.
3. An explanation of how the practice reduces waste.

A number of answers failed to provide the first part, but just jumped straight to named practice and talked only about waste in connection to that practice.

Other answers gave the first two parts of the answer, but failed to explain how the named practice reduced waste. (It was common to name a practice and then give a general explanation of what the practice was, but without referring to waste during this explanation.)

Finally, some answers even failed to name any practices at all!

One repeated cause of lost marks was that a number of students only stated that the named practice was related to reducing waste, without actually explaining how that reduction came about as a result of the practice. For example, the following sentence states that “self-organising teams” reduces waste, but does not explain how it does so:

The practice of self-organising teams allows developers to take key decisions for themselves in a highly organised and effective environment that minimises waste.

What is missing here is any explanation of how the practice directly results in the reduction of waste. Compare with this answer:

The practice of self-organising teams allows developers to take key decisions for themselves without needing to provide documents, e-mails and presentations needed for management to make these same decisions for them.

The first answer would earn 1 mark, for naming a relevant practice, while the second earns 2 marks for naming a relevant practice and explaining how waste is reduced.

b) Most candidates answered this short question correctly. The answer, of course, was to deliver value to the customer. I gave a mark for any answer that referred to some concrete, useful end for the customer.

Half marks were awarded for answers that described things the customer would probably want, but which were a means to an end, rather than a useful end in themselves. Typical answers in this category were: “working software” and “embracing change”. These are good things, but only in so far as they tend to help us deliver value in the end.
Q2. This was the most popular of the questions in section B. Almost every candidate answered it. That turned out to be a good choice, as most people achieved high marks in it.

a) Answers to this part of the question were mixed. Several of the values applied in each case, to differing degrees, and I awarded marks wherever a sensible connection was made. The main cause of lost marks was in considering only one side of the value given as the answer instead of both of them. For example, in scenario iii), the team is not valuing “customer collaboration”, because they are busy turning their customer into another developer. But, there is nothing in the scenario to suggest that they are putting high value on “contract negotiation”. Therefore, high marks could be awarded for picking this value for this scenario only if an explicit link to contract negotiation was made in the question.

Other marks were lost through not reading the scenario descriptions carefully. For example, a number of candidates complained that the team in scenario i) would not be able to create any software until the document had been reviewed. But, the scenario states that the requirements doc is being created for the next release. The team is still working with short iterations, and short incremental releases. This means they can get on with generating working code, while waiting for the requirements review for the next release to be finalised.

b) This question was in general answered very well. You all seem to have got the hang of using task board columns to represent key stages in the development that need to be tracked, to inform team members of when tasks have arisen that need their attention.

c) There were some very sensible suggestions for new retrospective structures, although few candidates seemed to want to give actual prompts (as opposed to lists of “topics that could be discussed”). Opinion was divided over whether holding retrospectives in the pub was a good idea or not in general, but most student agreed that this team needed to retreat to a more formal setting for their retrospectives, at least until the issues with the team had been resolved.

Lots of answers stated that team members should be told to think about why new members were not wanting to stay in the team and why customers did not request the team again. This much is obvious - and is basically what the question says. The point of the question (and what made this a suitable end-of-question question) was to work out how to get the team to address these issues without being confrontational about it or making the team feel attacked. There were a few good suggestions for achieving that, though not very many.

Q3. This was the second most popular question from section B. There were many excellent answers, but overall the marks achieved were more variable than for question 1.

a) There were good answers to this question, although many candidates earned only half the available marks for each sub-question by making only one substantive point in each case, instead of the two points needed to earn the full marks.
For sub-question (i), the point here was that the customer seemed to be asking the team to manufacture story point estimates to fit what they hoped would be delivered by a fixed deadline. This is not how story points work (on various levels).

For sub-question (ii), the point was that the team had carefully worked with the customer to order the stories by value, but this particular developer was ignoring that in order to work on a more technically interesting story. The developer was confusing “technically complex” with “of value to the customer”. Sometimes the most valuable thing is a simple report that gets the information needed to the right person.

A small number of people seemed confused by my use of the term “report” here, thinking that it referred to a document created by the project team describing the progress of the development. I awarded some marks for these answers where they were accompanied by a convincing diagnosis of the problem, in terms of fundamental misuse of the task board by the team.

Thanks to the person who pointed out in their answer that no real developer enjoys working with REST interfaces! :-)

For sub-question (iii), the point here was that the developer in question had misunderstood both the role of tests in an agile project, and the role of an independent test team in general. A number of candidates claimed that, since the team was using agile methods, they must be using TDD. This is not the case. Not all agile teams use all agile practices. It’s perfectly possible to be a highly agile team, and not to use TDD. I would expect even the most agile teams to use TDD selectively, in any case, depending on the specific features of the code being developed.

In general, when you hear statements that an agile team must do this or must do that, the statement will be incorrect. The underlying principle behind agile approaches is not to slavishly follow any one approach or maxim, but to apply the concepts and techniques only in so far as they help the team to progress effectively.

Whenever I set questions involving the use of an independent test team on an agile project, some candidates get all huffy about how ridiculous that is, because a truly agile team will be taking responsibility for all testing themselves. Independent testing can still provide enormous value for an agile team, since a fresh viewpoint is always valuable in finding the errors that the development team is too close to see. The mistake would be in handing over all responsibility for testing to an independent testing team (especially for unit testing, which isn’t something an independent testing team would normally be expected to be doing).

b) This question was answered well, with some thoughtful justifications. I awarded marks for all sensibly justified viewpoints, meaning that answers giving completely the opposite viewpoint on a particular story were both awarded marks. Most causes of lost marks resulted from a failure to note the details I had carefully placed in the questions. For example, while a general reminder of debts to the University is probably useful for students, it is questionable as to whether a weekly reminder of something that only really has an impact on the student at the end of the academic year is really delivering value to this user role.

c) This was perhaps the question that was answered least well by candidates across the whole of the exam paper, with many candidates failing to earn any marks at all.
The question asked you to add a business value to the user stories given in the preceding question. Many candidates interpreted this as asking them to waffle vaguely about the possible business values of the stories (despite having already covered this ground in the previous question). Since plenty of candidates had managed to interpret the question correctly, I awarded marks only for answers that took the form of a business value from a user story.

Those that did interpret the question correctly provided some excellent business values, and, in general, earned high marks.

d) As should perhaps be expected of the final sub-part of a question at 3rd year level, this question proved challenging, with very few really satisfactory answers. The challenge here was to turn a vague and abstract epic into a sequence of stories that could clearly be implemented, using information sources that could be expected to be available. Many candidates ended up with solutions that kept the epic behaviour intact but added additional stories around it. For example, a number of candidates gave some preliminary stories about uploading and storing CVs, following by a story that said that CVs should be ranked by participation in hackathons, but without giving any detail as to how this was to be achieved - just like in the original epic.

Only a few students considered the question of where the information about hackathon participation could be found. Some suggested that students should enter this information in a structured form, when uploading their CV - which was sensible. Others sketched out methods whereby CVs would be parsed and analysed, in an attempt to extract information about hackathon participation and success. Given a list of current and past hackathons, this should be relatively easy to do. Others assumed that some HR staff member would eyeball the CVs and enter the data that way - which was not the best approach, but was still miles more convincing as a solution that the candidates who assumed that the information would just appear as a result of writing a story about it.

Q4. This was the third most popular question from section B. Almost everyone made a good attempt at designing the test table, with the fixture code also being correctly created in most cases.

a) This question was answered well by many candidate, but a significant number of candidates seemed not to be able to pick out the key details from the question, and so gave inappropriate or irrelevant answers. The key detail was that the customer’s previous experience was that the post-it notes contained what she wanted, but the software that resulted from it seemed unconnected. Full marks were awarded only to suggestions of practices that focussed on diagnosing and avoiding this problem.

As usual, there were a number of answers that clearly came from candidates who had stopped attending the lectures early in the semester, and who struggled to name any agile practices correctly (“iterative and incremental agile development” is not a practice I am aware of …). There were the usual handful of answers where candidates had confused agile methodologies with agile practices. I did not award marks for any suggestions that “XP” was a “practice” that this team could adopt.

b) There were many perfect answers to this question. The main cause of lost marks (and it was only a single mark in most cases) was failing to include the “basic prices” as an
input in the table (or in some other form). I put a big clue to the fact that these should be included in the test (and not assumed to be £10 and £5) in the conditions of satisfaction on the user story. Here, I noted that the basic prices change regularly. In this case, test designs that assume that the basic price is fixed are not appropriate.

The examples themselves were very easy to produce from this scenario, and most candidates had no problem producing 8 non-redundant examples. In the few cases where marks were lost in this part of the question, it was through not interpreting the rules for calculation of the final prices correctly.

A small number of candidates thought hard about the story and decided that the examples should be about whether the prices met government regulations or not, rather than the actual calculation of the prices. I accept that there was an ambiguity in the story itself on this point, and so awarded marks for these answers where I could. The fact that so large a part of the conditions of satisfaction were concerned with the correct calculation of price was a big clue as to the intended meaning however, so I only awarded full marks where the correctness of the calculation was in some way included in the table design.

c)

In general, this question was answered well by most candidates. Almost everyone remembered the basic structure of a decision table fixture class, so the first 4 marks were achieved by almost everyone. The second 4 marks were for the programming-by-wishful-thinking element of the fixture, and the marks here were more variable. Many candidates produced sensible designs, but some answers relied too much on a small number of rather artificial seeming static methods. Since this was the last sub-question, it was intended to distinguish borderline first class students from students who are amongst the very top in the year. As such, I awarded full marks only for designs that made sense both as production code and in the context of this test.

While marking, I was very pleased to find that everyone who attempted the question had a clear understanding of the difference between test code and production code, and no one attempted to actually implement the price calculation in the fixture code—until, that is, I came to the last two candidates who had attempted this question… Maybe next year I’ll manage to get the whole class understanding the difference between these two types of code!

Q5. Only a few candidates answered this question, but almost all received high marks.

a) Candidates had no difficulty with this part of the question.

b) Answers to this part were more variable. Most candidates were able to say that they would write a (single) failing test as the first action. No one fell into the trap of saying they would write lots of unit tests and then code up the production code (which would have been TFD, not TDD). But very few candidates gave a sensible justification for why they had chosen to start with the particular test they had written down.

c) This part was for the most part well answered. Almost all candidates seemed to have a good grasp of the RGG-RGG-RGG sequence of coding steps that the question asked for. While the test cases were rather longer than the one in my sample solution, everyone was able to navigate sensibly through the path of test possibilities, and some answers discussed the effect of the order of tests on drawing out the patterns for refactoring very convincingly.
The coding and refactoring steps were less well carried out. Few candidates, it seemed, could bring themselves to actually write the minimum amount of code needed to make the tests pass in the early cycles of the TDD process. And the most obvious and useful refactoring options were often not spotted (such as dealing with literals in the code and the use of a Map data structure to represent the conversion table equivalences), while some of the refactorings that were applied served only to lengthen and obfuscate the code, as far as I could see.

This was the first year in which no one attempted to answer this question who clearly had no idea what TDD was. No one this year treated the question like a COMP16121 question, and simply tried to write the production code directly, with no tests.