Feedback on Student Performance in the COMP33711 January 2014 Exam

Suzanne M. Embury

Section A

Q1. This compulsory question was answered very well by many candidates, although others lost marks through poor exam technique. (In fact, this statement is true of the exam as a whole.) Where marks were lost, it was for:

- Failing to actually name any agile practice in the answer.
- Naming an agile value instead of an agile practice.
- Naming an agile practice but failing to provide any justification for its connection to the given principle.
- Providing a justification that simply repeated the words from the question, without adding anything new.
- Stating that there is a relationship to the given principle, but not actually explaining what that relationship is.
- Providing an unconvincing or irrelevant justification.

a) A number of candidates provided a justification for their selected practice in terms of “frequent delivery” of software, but failed to also justify that “value” was being delivered. A common mistake was to talk about the delivery of working software, but “working” does not mean the same as “valuable” in this context.

b) A significant number of candidates stated that stories/taskboards/etc made it easier to change requirements, but a much smaller number actually explained the features of these practices that brought this about.

c) This part was answered well.

d) Several candidates claimed that learning spikes contributed to technical excellence and design quality, perhaps misinterpreting this principle as a reference to the skills of the developers rather than the excellence of the software. None of these answers came along with a very convincing justification. I was also surprised by the number of answers that included claims that pair programming is “fast” and “cheap” compared to individual developers working alone.

e) A disappointing number of students mistakenly named agile planning practices, such as stand-up meetings and iteration planning meetings, as opportunities for reflection.

Section B

Q2. This was the most popular question, being answered by around 2/3 of the class.
a) There were many excellent answers to this question, although weaker answers displayed only a very superficial (and often mistaken) understanding of BUFR. For example, a number of candidates claimed that BUFR always led to wasted time at the start of the project, and that BUFR approaches allowed no interaction between the development team and the real customer.

As usual, marks were also lost through poor exam technique. A number of candidates talked about the scenario in general, and forgot to relate their points specifically to the use of BUFR. Others talked about BUFR in general, and failed to link their points to the detail of this specific scenario. Some candidates failed to give a clear and explicit statement of their confidence (or lack of confidence) in the team’s ability to deliver value by the deadline.

The fact that scenario iii) was divided into several year long stages seemed to confuse a number of students into thinking that this was an agile project. The project was incremental, since it was big and ambitious and the company knew it needed some way to control the risks on it. But being incremental is not the same as being agile. 1 year is far too long for an iteration in an agile project; finding out you’ve got it wrong after 11 months of work hardly counts as “failing fast”.

The description of scenario iii) was not very clear as to whether requirements gathering would be performed once at the beginning of the whole five years, or at the beginning of each year long stage. I intended the latter, but accepted either assumption in answers.

b) Most candidates earned full marks on this question. The only roles suggested that did not earn marks were: “customer of the bank” and “competitor organisations”. I’d have needed to be given a really solid reason for these roles to be accessing the software before I could offer marks for them.

c) Yet again, poor exam technique claimed some marks in this question, when candidates failed to indicate which role their answer referred to. Others named a role but failed to specify the type of business value they were aiming for in their answer. However, the main cause of lost marks here was in providing vague, incomplete or unconvincing descriptions of the business value, and/or of the route to achieving it.

Q3. This was the second-most popular question, being answered by just under 2/3 of the class.

a) There were very few really excellent answers to this question. Many candidates seem to have ignored the description of the team’s process, given in the question, and to have given some standard columns from a typical taskboard, whether they were right for this team or not. For
example, very few people created a column for stories that were waiting to have their acceptance tests written and automated (an “in specification” column), ahead of the main “in development” column. A number of people included columns to say that acceptance tests should be run against the code, after its story leaves the “in dev” column. This makes no sense if the team have automated acceptance tests, which would be run continually during development, anyway.

Other frequent omissions were columns to help the team handle releases at the start of the process, plus the exploratory testing and automated build testing steps at the end.

b) Some really excellent stories were produced in answer to this question. Marks were lost for stories that had a poor business value (for example, many stories had business values that were in fact functions of other stories), that were not end-to-end, and that were too large in scope for a 2-week iteration.

c) Many candidates produced appropriate epics. Marks were lost for epics that were just too vague (“I want to handle orders”) or were very concrete and small. A small number of students resorted to stringing a collection of tiny functions together using “and”, which obviously didn’t earn marks. One rather cheeky answer involved combining the three stories from part b) in this manner, in an attempt to create an epic. I was most impressed, however, by the fine exam technique displayed by one candidate, who managed to create a single epic story covering the entire application! There was no danger of this story being classed as “too small”.

d) This question was not answered well by many candidates, although those answers that were good were insightful and interesting. The main problem was that too many candidates wanted to talk about how the code could be tested, when the kitchen team hadn’t yet provided the dependent components, whereas the question asked how they could “deliver real value”, meaning that the software delivered had to be capable of running live. This rules out the use of hard-coded values, and stubs, mocks and simulations. Our only option is a sophisticated fake, as some candidates suggested, to dovetail the new processes used by the front of house staff with the old processes still in use in the kitchen. The most promising suggestion, IMO, was to print out the orders, and pass these through to the kitchen staff, as in the current manual system.

A handful of candidates said that if there were technical dependencies between the stories, then the teams had not written them correctly (to make them fit the INVEST criteria). If all the work was being done by one team, this would definitely be correct. But the question talks about two separate teams. Unfortunately, it is not so easy to avoid technical dependencies between stories between teams. In a single team, we can schedule the stories in any order we like, and implement each technical
component as we need it. We don’t have this same control, if a technical component on which a story depends is to be built by another team.

Q4. This was the third most popular question, being answered by around a third of candidates. The performance was very diverse, however; this question had the highest mark (19.5) and the lowest mark (3.5) of all the questions in Section B).

a) Most candidates were able to answer this question well. Weaker answers talked about UML and BUFR in general, rather than concentrating on issues specifically relating to the maintenance of UML specifications. Some students seem to be under the mistaken impression that UML diagrams are self-explanatory, and regarded the existence of the accompanying documentation as indicating a fault in the team’s use of the notation.

b) There were very few good answers to this question. Many candidates seemed to have only a weak grasp of what specification-by-example involved, talking about task-boards and user stories, rather than examples. Others stated that a specification-by-example was easier to maintain than a UML one, without actually explaining why that should be the case. Many of these were wildly optimistic about the benefits of specification-by-example in terms of requirements change, but most answers seemed to assume that adding new requirements was the only change that needed to be taken into account. It was great, however, to see the answers from students who were confident enough with the idea to point out weaknesses in the technique, and to provide arguments that specification-by-example wouldn’t help the team resolve the problem they had identified.

c) There were many excellent table designs, and test cases, given in answer to this question, although only a handful of candidates managed to get the most efficient solution (spotting that the length of time on the contract actually wasn’t at all relevant for this story). I did not penalise students who included this column in an appropriate way, but did award bonus marks for those who saw that it wasn’t needed.

The most common loss of marks was in misinterpreting the story that the test was for. The story asked specifically for the number of txts the customer could send that month before the price per txt would rise. However, some candidates were unable to resist including information in the test about how much the txts would cost in that new price band, how much the txts sent this month had cost, what price bands had been applied in the current month, and how much the total costs of all txts sent by the customer would be. None of these quantities were required for the specific story as written. If this kind of information is important to achieve the required business value, then other stories will exist that will
indicate that it should be fetched. We don’t have to worry about it in this test.

For the test cases themselves, marks were lost for failing to match the business rules given, covering redundant cases, and choosing random values for cells that do not indicate the important boundaries mentioned in the business rules.

d) I was pleasantly surprised by the excellent answers given to this question, and especially by the sensible design choices made while programming-by-wishful-thinking. Few people included appropriate fakes, but this was worth only a tiny fraction of the marks, and some came up with designs that genuinely did not require a fake (in which case, of course, they were not penalised).

The marks were split into 4 marks for the fixture structure and 4 marks for the design produced by wishful-thinking. For the fixture structure, the most common cause of lost marks was in failing to link the computation of the output value (or values) to the input values. For the design, the most common causes of lost marks were missing elements from the design, and awkward responsibility assignment (such as designs that meant customers could have only one contract at a time, by putting all the work on the Customer object).

A small but worrying number of students actually wrote production code (i.e., code to compute the number of txts till the next price rise for a general customer) inside the fixture class. This reflects a serious misconception as to what the role of the fixture class is, and meant that no marks could be given for the design element of the question. I can only advise that candidates who made this mistake quickly learn the difference between test code and production code, as this is now a really basic concept in the way code is produced in industry.

Q5. This was (by a small margin) the least popular question, being answered by just under a third of students. It was great to see that a significant number of students have a good working understanding of TDD, and were able to earn very respectable marks. Unfortunately, some candidates chose to answer this question without apparently having any understanding of TDD at all; naturally, those answers earned very few marks.

a) This question was answered well in general, although (again) some candidates failed to read the question carefully and lost marks. In particular, some students seemed to be determined to answer the questions I had asked on the sample exam, even if the real exam was asking for something quite different. There were therefore a number of candidates who interpreted this question as asking: are these people doing TDD correctly, rather than asking what their next action should be.
This resulted in a number of answers that did not contain a specification of an action at all, and I could not award marks for them. Other marks were lost by candidates who did not understand the distinction between tests failing in the ATDD process and tests failing in the TDD process (for example, assuming that all failing tests must be fixed immediately, even if they actually represent new requirements for non-urgent stories).

b) Many people were able to point out the production code step that follows logically from the failing test the programming partner has provided. The second step proved more variable. The correct answer was to undertake a refactoring step (and to describe what, if any, refactorings would be performed). The most common (significant) mistake was instead to give another coding step, in which the full production code for the functionality required was given, in clear contravention of the basic TDD rule, that production code can only be written in response to a failing test.

c) This question was answered very well by a large proportion of candidates. The majority of answers gave two sensible tests, sensibly expressed in correct JUnit. The most common cause of lost marks was for tests that did not exercise the ObservationSet.longestObserved() method, but instead tested some other random functionality, not mentioned in the question at all. For example, some candidates wrote tests concerned with measuring the number of items in an observation set, or checking their validity.

Many candidates with otherwise perfect answers lost a mark by failing to spot that the requirements for the code state that the only allowable character in the observed strings is “A”.

d) There were only a few really good answers to this question, although many candidates managed to earn some marks for it. The problem seemed to be that a significant number of students chose to interpret this question as if it were asking “how agile is this team?” and therefore suggested that random agile practices should be adopted, rather than focussing specifically on readiness to adopt TDD. Some agile practices are dependent on others, but many can be adopted independently. TDD, for example, does not need pair programming to succeed (although it works well done in pairs). Nor does it require practices such as user stories, task boards or specification-by-example. Answers which suggested these earned no marks (although I would have been prepared to award marks if a really great justification had been provided, rather than simply listing the practices without explanation of their relevance to TDD).

Another significant cause of lost marks on this question was for answers which said that the team was not TDD ready because they were not currently doing TDD! This circular logic was not convincing and so did not earn marks. The only other common mistake made was to focus all the corrective actions on testers: testers should learn to work with developers to write code, testers should learn to write automate tests,
testers should learn to be developers, basically. But TDD is an **internal** code quality practice. It is the job of developers. While it is very useful if the testers on a test can code and understand TDD, the people who really need to be able to apply it are developers. So, answers which focussed on how the testers should change, without addressing the developers, earned few marks.