Marking Scheme for Sample COMP33711 Exam Paper

January 2018

Suzanne M. Embury
Christos Kotselidis

The exam format for this course unit has changed for this year, and now consists of just 2 compulsory questions, both worth 25 marks. The first question covers material on agile processes (roughly the material covered in weeks 1 to 5), while the second covers material on technical agile practices (roughly the material covered in week 7 to 12).

You should answer each question in a separate answer book. This is because each question is marked by a different person. Using separate answer books for each question allows us to mark the exam in parallel.

The sample exam follows the same structure, so you can get an idea of what to expect on the day.

This marking scheme describes how we would mark the answers to the sample exam, as well as providing model answers.

Question 1: set and marked by Christos Kotselidis

a) This question aims to test your understanding of both the basic agile values and some of the key practices we looked at in the course unit, by asking you to match up the practices against the value.

For each practice, I'll award 1 mark if it is matched with a plausible value, and one or more further marks for a good justification for the match, up to a maximum of 10 for the whole question. Good justifications will demonstrate a deep understanding of the concepts underlying both the value and the practice. If the same point is repeatedly made with regard to two or more of the practices, then that point will only be marked once.

For example, if a student matched every practice against value a), on the grounds that all the practices were carried out by people interacting, then than point would earn 1 mark the first time it was made, and 0 marks afterwards.

If no justification is given, just a value, then I'll have to make a judgement call as to whether I think the value is self-evidently and obviously linked to the practice. Please help me take the randomness this inevitably introduces out of my marking by always remembering to provide a justification.

Model answers are given below. Note that is it possible to match the practices plausibly against other values than those noted here. Many different answers could earn full marks if sensibly justified.

i) The task board practice matches with value a). In conventional teams, complex planning tools and practices are often used, which are often accessed electronically through complex authorisations. The planning tool or process becomes more important than the individuals on the team. The task board, on the other hand, is designed to support productive interactions between team members: to focus on the decisions themselves, not on how those decisions are recorded.

ii) The practice of assessing when stories are "done done" (as opposed to just "done") matches with value b). In this practice, a story is considered complete only when working software can be run in the deployment environment. This is the key test, and not whether any documents or checklists have been completed.

iii) This practice matches with value d). Business-value-based release planning involves planning releases so that high business value stories are delivered early, and in sensible combinations that together deliver business value. If the needs of the customer changes (so that the notion of what is high business value changes) then this can be reflected by changing the position of the stories in the release plan. The practice is lightweight, so the overheads of change are low.

iv) The on-site customer practice matches value c). In this practice, the customer sits with the team, to help it make decisions about what to build and what not to build. There is no contract to bind the two parties, and instead the customer is expected to work with the team as an equal team member, to make sure that useful software is built.

v) Pair programming matches with value a). It ensures high quality code by creating a situation in which two developers can interact continually, to both find problems and correct them quickly and informally. This is in contrast with the heavyweight code quality processes used on conventional projects (such as formal code reviews).

I've written the answers out fully here, so that they are comprehensible in this form. In the exam, you wouldn't need to write out the practice name and could just say:

i) Value a) because…

b) i) Roles clearly implied by the scenario are: location predictor, route planner, toy maker, toy deliverer, field worker. Toy manufacturing planner might also be considered to be implied, and child can be considered, if it is clear that there is some way in which they access the software, or otherwise have some major impact on business value.

1 mark to be awarded per sensible role with a satisfactory explanation. Just naming the role will award you 0.5 marks.

ii) 1 mark for each story that is clearly supported by the description of the scenario and that fits its description as either an epic or a single iteration story, plus an additional 1 mark for stories that correctly use the basic template, up to a total of 6 marks. Examples of acceptable stories are:

- As a toy deliverer, I want to know the location of the child I should be visiting next, plus my scheduled arrival time, so that I can **ensure** that every child on the route is visited, by the delivery deadline.
  (A **single iteration** story.)

- As a location predictor, I want to simulate the movements of groups of children based on a model I enter, so that I can **quickly** estimate their probable location on the upcoming delivery date.
  (Note the use of "quickly" in the last part of the story: the goal becomes a "next step" function without this. This story sounds like an **epic** to me.)

- As a location predictor, I want to be warned if children deviate from their predicted movement pattern in the weeks coming up to the delivery date, so that I can **maximise the accuracy** of my predictions for the final delivery.
  (Note: this is another story where it is very tempting to just put the next step in the process as the goal. You need to push through to the actual business value to get it right.)

- As a field worker, I want to submit reports of good/bad behaviour incidents, so that **fair** toy selections can be made for the final delivery.

Obviously, candidates would only need to provide 3 user stories in their own answer to the exam. We give 4 stories here only to increase the number of examples available to students for revision.

iii) You can pick a user story from your answer to b) ii) or present a new adequate one. For example, let's use the following user story:

As a toy deliverer, I want to know the location of the child I should be visiting next, plus my scheduled arrival time, so that I can ***ensure*** *that every child on the route is visited, by the delivery deadline. (A* ***single iteration*** *story.)*

Let's assume that during planning poker the senior developer rated it with 8 story points while the junior developer rated it with 2 story points. In this example, the more senior developer believes that this user story is more complicated in comparison to the junior developer.

Upon revealing the poker cards the senior developer argues the following:

> "I believe that this is a complicated user story because we have never worked with integrating Maps API in our applications and therefore we have no expertise. Furthermore, for that user story we have to combine the different addresses with our route planning software and dynamically adapt it in case we face delays during the delivery."

The junior developer argues the following:

> "I have previously worked with the API and it provides route planning functionalities. However, it does not dynamically re-calculate or re-calibrate the planned route upon delays."

We see from the answers above that:

- The senior developer did not know that a member of the team has previous expertise with this API.
- The junior developer did not think about the fact that they would need to dynamically re-calculate the planned route in case of delays.

A middle solution would be 5 story points, which will leverage the expertise of the junior programmer, but factor in that more work has to be done rather than just integrating the existing Maps API. (1 mark per sensible point given in the answer, or less in case not enough justification is provided).

Questions 2: set and marked by Suzanne M. Embury

a) 2 marks per scenario: 1 mark for a plausible problem diagnosis that fits the given facts, and 1 mark for proposing a corrective action that stands some chance of resolving the problem diagnosed. Below I give some examples of the kinds of point that could be made. Obviously, for just 1-2 marks, I would not expect people to write as much as I have written here.

i) An agile team where testers regularly don't have enough to do is definitely not operating correctly. As we saw in the lectures, testing is central to agile and should be taking place right from the time that the first story is gathered. Even before this, infrastructure for testing and continuous integration can be being assembled. In this case, it seems that the team is not gathering test cases as central part of their requirements gathering; or perhaps they are not automating test cases as they are extracted from the customer/team? Perhaps the testers on this team are operating to the traditional view that there is nothing for them to do until a full requirements specification has been written, or some code has been implemented. Perhaps they are not acting as one of the main points of interaction with the customer? The solution is to get them involved in story writing and test case automation right at the start of each iteration. They probably need some training on specification-by-example and/or automation of acceptance tests. And the team as a whole perhaps needs some coaching in the role of testing in agile projects, and also on whole-team-responsibility.

ii) This team appears to be working in an organisation that has traditionally separated testing out into a different function from development, and has siloed its staff based on quite narrow technical roles. Probably, the experienced agile developers are using test-driven techniques of some kind and are writing a lot of acceptance and unit tests as a means of discovering and documenting the requirements. But they have not taken the time to coach their fellow team members in the new approach. And the team seems not to have worked out explicitly how it will work with the dedicated testing teams the organisation owns. Such teams can still be useful for an agile team's work, but they should be used in addition to and not instead of the normal automated testing work of the agile team itself. The obvious "solution" is to provide more training for the existing employees, but this may not help them to change their mindset. More powerful approaches would be to focus on defining the team's own quality goals and how they will meet them (perhaps in an uncoming retrospective), or to spread expertise and awareness throughout the team through careful use of pair programming.

iii) Despite claiming to be agile, the team seems to be locked into a traditional "requirements gathering then implementation" approach. There was lots of interaction with the customer initially, but the team seems to think they don't need to interact with the customer during coding. They are writing lots of tests, but they seem not to be checking that the tests they are writing actually do describe the behaviour that the customer will consider valuable. The result is inevitable: they are building the system they think the customer wants, instead of working continually with the customer to discover the best solution. Team training is always an obvious solution in these cases, but really the team needs to face up to the fact that they are building the wrong thing. One possible approach would be to shorten their iteration length, and to arrange highly visible showcases with multiple customer representatives at the end of each one. Introducing customer-facing testing techniques could also help, since the customer representative would be able to be more involved in checking that the tests match the vision for the product.

For full marks, the solution proposed should deal with any mindset problems in the team, as well as just addressing any missing skills. Generic answers saying that the team doesn't understand agile and needs to receive more training will only receive marks if backed up by justifications related to specific elements of the scenario in the question. Answers which could apply equally well to all three scenarios are unlikely to be specific enough to earn marks.

b) There are two types of discount mentioned and therefore we need to create two groups of scenarios to represent both of them. It is not necessary to provide full feature definitions in answer to this question (or any question in the actual exam on Cucumber). Just a sequence of Cucumber scenarios is all that is required, although candidates who do provide full features will not be penalised.

The shipping discount appears straightforward, and can be specified with just a few basic scenarios:

Given a customer who is a loyalty scheme member
And the customer places an order for products totalling £30.01
When the order is confirmed
Then the shipping costs for the order will be £0.00

Given a customer who is not a loyalty scheme member
And the customer places an order for products totalling £30.01
When the order is confirmed
Then the shipping costs for the order will be £3.00

Given a customer who is a loyalty scheme member
And the customer places an order for products totalling £30.00
When the order is confirmed
Then the shipping costs for the order will be £3.00

The question doesn't say what the shipping costs are or what they are based on. In this case, I assumed a simple single-cost model and made up a plausible value. In the actual exam, you should make similar decisions about aspects of the specification not made explicit in the exam. Any plausible approach will be considered correct. If in doubt, you can add a note explaining what you were confused about and what you have done about it in your answer.

The scenarios needed to express the "cheapest item free" discount could be:

Given a customer who is not a loyalty scheme member
And the customer orders a book costing £10.00
And the customer orders a pen costing £3.50
And the customer orders a pad costing £5.00
When the order is confirmed
Then the order total before shipping will be £18.50

Given a customer who is a loyalty scheme member
And the customer orders a book costing £10.00
And the customer orders a pen costing £3.50
And the customer orders a pad costing £5.00
When the order is confirmed
Then the order total before shipping will be £15.00

Given a customer who is a loyalty scheme member
And the customer orders a book costing £10.00
And the customer orders a pad costing £5.00
When the order is confirmed
Then the order total before shipping will be £15.00

Given a customer who is a loyalty scheme member
And the customer orders a book costing £10.00
And the customer orders a pencil costing £1.50
And the customer orders a pen costing £3.50
And the customer orders a pad costing £5.00
When the order is confirmed
Then the order total before shipping will be £18.50

Given a customer who is a loyalty scheme member
And the customer orders a book costing £10.00
And the customer orders a pencil costing £3.50
And the customer orders a pen costing £3.50
When the order is confirmed
Then the order total before shipping will be £13.50

Notice that the wording here and in the previous stories makes clear that this discount is to be applied before the shipping costs are calculated. Other solutions which are consistent with the wording in the question will also be marked as correct.

Marks will be awarded as follows:

- Up to 2 marks for broadly correct use of the Given-When-Then structure. For full marks, the “when” step should describe an action taken by the user through the software by some mechanism, and should not be another pre-condition for the scenario. The “given” and “then” steps (including “and” and “but” clauses) should describe conditions on the state of the world/system.

- Up to 2 marks for writing scenarios that are compatible with the behaviour described in the question.

- Up to 2 marks for writing scenarios that cover the key aspects of the required behaviour. Here, specifically, there must be scenarios that show when the discounts are not applied, as well as scenarios showing when it is. The scenarios should address the key ambiguity of whether the shipping discount is applied after the “cheapest free” discount. However, it is not expected that students will specify all aspects of the behaviour. In general, it will be enough to give a scenario for the simplest version of the discount, a scenario when the discount is not applied, and a scenario showing some other aspect of the behaviour (like what happens when two items have the same cheapest cost). It is not necessary to give exactly the same set of scenarios as given above (or as many) for full marks.

- Up to 2 marks for writing scenarios that read well, and that use customer oriented language to describe the required behaviour in a way that non-technical customers will recognise. Technical terms such as “button”, “dialogue box” and “menu” should be absent to earn these marks.

When answering questions that ask for Cucumber scenarios in the exam, you should feel free to make use of abbreviations and “ditto” marks to avoid having to rewrite scenarios out in full each time. For example, the version of the scenarios above is completely clear and acceptable:

Given a customer who is a loyalty scheme member
And the customer orders a book costing £10.00
And “ “ “ a pen “ £3.50
And “ “ “ a pad “ £5.00
When the order is confirmed
Then the order total before shipping will be £15.00

Longer words like “customer” can be abbreviated to “cust.” in later scenarios provided the word has been used in full in some previous scenario and it can’t be confused with some other key word used in earlier scenarios.

c) One step definition is needed for each distinct step used in the scenarios. This means that 4 step definition methods are needed for each discount option. Students are asked only to give step definitions for one of the two discounts, so 4 step definitions (two Given steps, a When step and a Then step) are required in the answer.

Marks should be awarded for the glue code as follows:

- 1 mark if all the steps in the selected scenarios are correctly matched by some glue code regular expression.
- 1 mark if all input values are matched and passed to the glue code correctly.
- 1 mark if the glue code for the When step invokes the production code and stores the result correctly.
- 1 mark if correct use is made of JUnit assertions in the glue code for the Then steps.

Marks should be awarded for evolutionary design/programming-by-wishful-thinking as follows:

- 1 mark for sensible domain class design (including names).
- 1 mark for sensible method design (including names, and the classes they are assumed to be the responsibility of)
- 1 mark if the classes are linked appropriately (which may be having no link at all, depending on the class design that is chosen)
- 1 mark for writing glue code that creates a sensible and complete fixture, including handling of mandatory attributes not mentioned in the scenarios.

The glue code needed for both discounts is given below (obviously, you would not need to give all this code in your answer, as step definitions for only one discount are requested in the question).

```
package uk.ac.manchester.cs.comp33711.sampleexam;

import static org.junit.Assert.assertFalse;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.core.Is.is;

import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

public class SampleExamSteps {

   private Customer customer = new Customer("Freda", "Smith");
    private Order order = new Order(customer);

   @Given("^a customer who is a loyalty scheme member$")
   public void a_customer_who_is_a_loyalty_scheme_member() {
      customer.joinLoyaltySchemeMember();
   }

   @Given("^a customer who is not a loyalty scheme member$")
   public void a_customer_who_is_not_a_loyalty_scheme_member() {
      assertFalse(customer.isLoyaltySchemeMember());
   }

   @Given("^the customer places an order for products totaling
           £(\\d+)\\.(\\d+)$")
   public void the_customer_places_an_order_for_products_totalling(
                int pounds, int pence) {
      double cost = pounds + pence/100;
      Product product = new Product("Book", cost);
      order.add(product);
   }

   @Given("^the customer orders a (\\w+) costing £(\\d+)\\.(\\d+)$")
   public void the_customer_orders_something_costing(
                String productName, int pounds, int pence) {
      double cost = pounds + pence/100;
      Product product = new Product(productName, cost);
      order.add(product);
   }

   @When("^the order is confirmed$")
   public void the_order_is_confirmed() throws Throwable {
      order.confirm();
   }

   @Then("^the shipping costs for the order will be £(\\d+)\\.(\\d+)$")
   public void the_shipping_costs_for_the_order_will_be(
                int pounds, int pence) {
      assertThat(order.shippingCosts(), is(pounds + pence/100));
   }

   @Then("^the order total before shipping will be £(\\d+)\\.(\\d+)$")
   public void the_order_total_before_shipping_will_be(
                int pounds, int pence) {
      assertThat(order.totalBeforeShipping(), is(pounds + pence/100));
   }

}
```

d) This question aims to test candidates deep understanding of the strengths and weaknesses of TDD as a technique, as well as of the role of agile practices within agile teams. It is an open-ended question, and full marks can be obtained for answers that argue strongly for TDD as well as for answers that argue against the use of TDD in this situation.

Marks will be awarded for each unique valid point made in the answer.

Here is a sample answer that would earn full marks, and which is against the use of TDD:

I would argue against the use of TDD. I am assuming that the project is due to be delivered in a short time scale (in 3 months) and that at least half the team does not have TDD experience.
TDD is known to reduce productivity in the short term and is a challenging technique to learn. With such a high proportion of the team new to TDD, even

> *with pair programming it is going to take a while for the ideas to bed in, and 3 months is just not long enough for that. The team needs to be working well and producing code from day 1 of the project. I would suggest the team increases the amount of up-front testing it does on this project, but not to use full TDD.*

Here is a sample answer that would earn full marks, and which is in favour of the use of TDD.

> *I would argue for the use of TDD. I am assuming that the project is quite long (a year in total, though with early releases within that time scale of course, since this is an agile team). I am also assuming that the complex business logic referred to is really crucial to the project, and that getting it right is the team's biggest challenge in the project.*
> *With at least one experienced TDDer per developer pair, the technique can be picked up by the rest of the team in the timescales needed by the project. TDD will help the team implement the complex logic with confidence and will save the team time in debugging that logic later in the project, which will compensate for the time lost early on in learning the technique. The project will benefit in the longer term from the good test coverage that comes with TDD.*

Finally, here is a sample answer that would earn just 1 mark when marked generously, because of the vagueness of the reasoning and the lack of added information. Obviously, you don't get marks for restating what is said in the question, or for giving information not requested by the question (like the definition of TDD).

> *I would argue for the use of TDD. TDD is when you write lots of tests alongside the implementation of the production code, and use the red-green-green cycle. This project has complex business logic that is needed for TDD. The team has experience so it can learn the technique. It was able to learn other agile techniques, so should now try this one. The project is important so it should be well tested and TDD will help with this.*

Only this last sentence provides any new, relevant information, so 1 mark can be awarded, but no more.