Comments

Please see the attached.
Overall, answers were very good.

Answer Question 1 and two other questions

Note: throughout this exam, the phrase “MELT project” refers to the project as a whole as discussed in the course, not just the part you implemented in the lab. Where the latter is meant, a phrase such as “your MELT implementation” is used.

Question 1

a). The Waterfall method of software development is now obsolete. State the main reason why it failed, and one other.

   The main reason is that it couldn’t cope with requirements change. [1] Second mark for any other valid reason, e.g. Analysis Paralysis. [1]

   This question was generally answered well, although a number of people lost  a mark by not giving the main reason above.

b). Give two examples of applications, not discussed in the course, where the amount of time spent on ceremony is likely to be more than 20% of the total for the project.

   Anything large with lots of different functionality, e.g. Blackboard. [1] Anything non trivial which is concurrent/reactive/safety critical, e.g. a nuclear power station control system. [1] Note that there are no marks for saying what Ceremony is, the question does not ask that, or for just reproducing what’s on the lecture slide on this topic.

   Most people got both marks for this part.

c). Briefly define the “boss test” and give an example not given in the course of a use case which would pass it.

   The boss test says that a use case is appropriate if when an end-user is performing it, they would be able to give a sensible answer if the boss comes along and asks what they’re doing. [1] Hence it should be non-trivial but also a specific task. For the examples above: Blackboard: e.g. Check submissions for exercise. Nuclear Reactor: Monitor reduction in power output.
A number of people gave an example which was in the lectures (i.e. the ship navigation one) otherwise this was answered well.

d). State what kind things appear on a design class diagram but don’t appear on a domain class diagram, and one other way in which the two kinds of diagrams differ. (2 marks)

Software-oriented things like types and visibilities. [1]. They differ in purpose – a domain CD is to help us understand the problem domain, a design CD is to document the software design (or other valid difference) [1].

A number of people fixated on different aspects of the first point and missed the second.

e). Give two examples of classes in your MELT implementation, one of which is a Pure Fabrication and one which is not. (2 marks)

UI classes are usually PFs, and they probably also had classes involved in storing data, e.g. DatabaseConnector [1] separate from the “business logic” classes which are generally domain-inspired e.g. Question. [1]

This was generally answered fine.

f). Suppose you are implementing a CAD system for designing large buildings. Such a building will have a very large number of components, which will need to be represented as objects. Give two reasons why it would probably be a good idea to create these objects from a factory. (2 marks)

Since there will be lots of different components, some of which will be complex, the object creation logic will be complex, so it makes sense to encapsulate this in a factory (this means the code using components be able to create them using a specification, e.g. size 12 window rather than needing to know the exact constructor call needed.) [1] Secondly since there are likely to be lots of size 12 windows, the factory may be able to cache the objects representing these and save a lot of memory. [1]

Most people got the first point, but very few the second.

g). Give one advantage and one limitation of unit testing Java code with JUnit (2 marks)

Advantage: we can test the core classes thoroughly, giving good confidence that they are largely bug free. [1] Limitation: it’s difficult to apply to some parts of the code, e.g. UIs. [1]
Most people got the first point, but very few the second.

h). A successful test is one which causes the software to fail, but this is difficult for programmers to do with code they have written. State two different approaches to testing which aim to get around this problem. (2 marks)

Traditional approach – have a separate testing team, whose job is to be as nasty to the software as possible. [1]. Agile approach: write the tests before the code. [1]

I needed specifically both of these points, but a lot of people clearly didn’t know this.

i). Briefly state the concept underlying Design by Contract and how it relates to formal methods of software development. (2 marks)

DbC defines a “contact” between a method and its caller, analogous to a legal contract “if you provide parameters as specified in the precondition, I promise to provide the service specified in the postcondition.” [1] It is a “semi-formal” method, which at minimum provides useful documentation, without the overheads of fully formal methods. [1]

Most people got the first part, but not the second (maybe because it was not spelled out in the lecture notes).

j). Give one advantage, and one potential disadvantage of aspects as used AspectJ. (2 marks)

Advantage: localises cross-cutting concerns, e.g. logging. [1] Disadvantage: there is no indication in the base code that an aspect applies, which may lead to nasty surprises for the base code programmers. [1]

Answers were very mixed, depending or whether people knew about aspects or not

Question 2

a) The four key principles of the Agile Manifesto can be considered ethical principles. For each, state how; (4 marks)

This is bookwork, but requires a fair amount of knowledge.
The AM states: value:

1. People and interactions over processes and tools
   This is a very strong statement as Agile processes are quite prescriptive and tools are very important –still people and interactions are more important. [1]
2. Working software over comprehensive documentation (or minimise ceremony)  
Older methods prioritised documentation, even when this was clearly ineffective – ethically, it’s more important to actually deliver software! [1]

3. Customer collaboration over contract negotiation  
Fulfilling a contract without delivering what the customer needs is unethical.

4. Responding to change over following a plan  
As is following a plan when it diverges with what’s needed to deliver the best product. [1]

*This was generally answered very badly, although it was spelled out in the lectures.*

b) State four different groups of stakeholders, other than students, in a real-life version of MELT project, and what their principal concerns will be. (4 marks)

e.g.  
University Management will be concerned that the software is delivered on time and will provide value for money [1]  
The Exams office will be concerned that tests will fit in with their existing procedures (The tests are done under exam conditions [1]  
The English Language centre will be concerned that the software will enable them to test students effectively. [1]  
Technical support will be concerned about how easily the system can be deployed and managed. [1]

Note that it’s not enough to cite obvious functional requirements, e.g. “Question setters will be able to set questions” This kind of thing is very important in real life - if stakeholders’ concerns are not addressed adequately, all sorts of problems occur.

*Answers to this were mixed, but mostly ok.*

c). Give six rules (things to do or not do) in writing a UC, using Take Test use case for MELT for examples as appropriate [6 marks]

*Any good 6, e.g.  
Make sure all important steps (e.g. instructions before the clock starts, unlike BB) are included. [1]  
Make sure steps are given in the right order. [1] (Consider not including them if there is no “right order” e.g. details of question answering)  
Don’t go into more detail than necessary, unless the UC will turn into a user manual, which for Take Test it won’t because that will have to be nice and simple. [1]*
Avoid UI specifics, e.g. dialogs, ratio buttons – need to abstract away from them. [1] Non-functional requirements (e.g. answers are saved) can be documented along with it but not part of it. [1] Avoid technical language in general – the UC should be comprehensible to the users so they can comment on it. [1]

Note: the students are expected to know the requirements for the MELT case study, as these have been discussed at length in the lectures. The question is stated in this form rather than asking the students to write a UC, as experience shows that marking UCs objectively is difficult and time consuming.

Many people failed to find 6 distinct points.

d). For each of the 6 types of requirements in the FURPS+ checklist, briefly state what they are, and for each give an example for a Computer Aided Design system used to define civil aircraft. (6 marks)

Functional – what do users need to be able to do. E.g. the designer should be able to specify any available part in the inventory. [1] Usability (e.g. accessibility). Eg. The specs for the parts should be displayable in large fonts for designers with visual disabilities. [1] Reliability (e.g. dealing with hardware failure.) E.g. the system may fail to back up a current design no more than one time out of 100,000. [1] Performance (bandwidth and response time). E.g. Once a design session starts, display must be fast enough that no designer experiences noticeable pauses. [1] Supportability (ability to adapt to changing circumstances, e.g. different operating systems). E.g. the design software must run on Windows, Linux and Mac platforms. (in practice such software is often platform-specific) [1] + (other things, e.g. legal, packaging) E.g. the software must be displayed with the company logo on the initial screen. [1]

Many people failed to remember all of FURPS+ and hence got partial marks, otherwise ok.

Question 3

a). How can UML class diagrams be used for communication between software developers? Your answer should mention at least two different uses. (4 marks)

All UML diagrams are ceremony, and hence optional - it would be un-agile to spend a large amount of time on diagramming. [1]. Domain class diagrams can be used to help understand the domain, particularly for developers new to the project [1] Design class diagrams can be used to discuss design choices [1] UML diagrams can be used at the start of a sprint to help determine architectures choices for that sprint. [1]
Marks for anything else sensible, e.g. they can be used at any point to clarify the team’s understanding of the domain, design choices etc.

**In practice, most answers relied on experienced vs. inexperienced developer, and in practice both were needed for full marks.**

b). Explain, using an example from MELT, how a domain class diagram can be used to gather useful information from stakeholders. Your answer should take into account the different kinds of skills which different stakeholders have.

(5 marks)

An initial domain class diagram can be drawn after an initial interview with a stakeholder, to identify important domain concepts and understand the stakeholder’s language.[1] The diagram can then be inspected to identify area of uncertainly and generate further questions for the stakeholder (multiplicities are a particularly good source of questions [1]). An example from MELT is that there is a requirement for two or more invigilators per room, not per test (any other valid example is fine). [1] For stakeholders with an appropriate technical background, it may be most efficient to show them the diagram with areas of interest marked [1]. Otherwise the diagram should not be shown, but a list of questions arising from it should be prepared in advance. [1]

This was answered well, especially the part about stakeholders with different levels of experience.

c). Draw a domain class diagram which shows the following information about the representation of rooms in a university-wide timetabling system:

“The there are three kinds of rooms to be considered: standard teaching rooms, which can be used for any class which does not require specialist equipment, computer labs, used for all classes requiring computers, and specialist rooms, containing specialist equipment. In the last case, the kind(s) of specialist equipment are recorded, but not in detail because it is assumed that these rooms will only be used for classes in the school which contains them, e.g. only Chemistry students will use Chemistry labs. For every room, the school containing it, and the location within the school is also recorded. Also, it’s important to know the capacity (number of seats) of every room.”

Hint: do not add information not explicit in the above description. (5 marks)
Fully correct solutions should look very similar to the above. Note that the attributes schools location and capacity are common to all types of room and are therefore in the Room class. Deduct marks for all the usual sins,

- Using the wrong notation (e.g. wrong kinds of arrows, or putting multiplicities on the inheritance arrows).
- Putting types, visibilities in (this is a domain class diagram)
- Putting information in not justified by the description (e.g. how rooms are specified)

A fair few marks were deducted for these sins, particularly the last, although in practice many diagram were good.

d). State the fundamental rule which determines whether a use of inheritance is appropriate, and briefly explain why it is important to obey this rule. (2 marks)

Fundamental rule: only use inheritance if there is a clear is-a-kind-of relationship between superclass and subclasses [1] Important because if inheritance is used for other purposes, it introduces unnecessary coupling between each subclass and its “superclass” (or other clear explanation) [1]
Most people knew this.
e). Give an example of inheritance not given in the course, which does obey the rule, and another which does not. (2 marks)
OK: Car extends Roadvehicle (is-a-kind of relationship)
Not OK: Finger extends Hand (contains relationship)
1 mark for any valid example of each, not given in the course

There were a number of strange, although strictly speaking correct, answers to this.

f). Java has a very different approach to inheritance compared to other common OO languages, with only single inheritance (with the exception of interfaces, which have very limited functionality). Common “mixin” cases are dealt with by having some functionality (e.g. synchronization) built in. State whether in you view, this restricts the scope of a Java programmer in practice, briefly explaining why or why not. (2 marks)

Marks for any sensible statement in either direction. My own view is an unequivocal no – I can’t remember ever wanting to use MI, and I’ve seen many examples of it which don’t pass the is-as test and are therefore bad (there’s even one in the GoF book!). It seems that nature abhors MI! However, if they argue coherently in the other direction, they’ll get the marks.

Most people sided with me. To argue the other way would require a good example. I asked the same question on my third year course, with more mixed results, but no good examples of MI.

Question 4

a). Briefly explain what we should always aim for in terms of Coupling and Cohesion and give an example of each from your MELT implementation. (3 marks)

We should always aim for high cohesion and low coupling. [1] (Amazingly, most of them did not know this when they stared the course – I sometimes wonder what they were taught in their UG degrees!) 1 mark for a valid example of each from MELT.

b) Explain the GRASP principles of Polymorphism and Protected Variations, and how they are related, using examples from MELT. (6 marks)
Polymorphism – being able to send the same message to different types of objects and have them respond appropriately (implemented via inheritance). [1] Protected variations – abstract the Thing Which Varies, so when the Thing Which Varies varies, it doesn’t trash the rest of the code. [1] Often achieved via polymorphism, as having an abstract class decouples the rest of the code from the concrete subclasses [1] e.g. in MELT it allows us to add new question types as subclasses of Question without changing existing code. [1] PV can be achieved without Polymorphism, e.g. choosing fonts which will work across platforms. [1] Polymorphism only achieves PV if it is used wisely however (e.g. used to model roles, and in many examples from MELT implementations it has the opposite effect) [1] Other sensible points also get marks.

Answers to this were very mixed – few got mull marks.

c) Explain the notion of a Controller, and the different types of controller, using an example of each kind. (3 marks)

A controller is the first object beyond the GUI layer which controls system operations.[1] A façade controller represents the overall “system” or “root object”, or a specialised physical device. This could be used e.g. in a cash machine. [1] A use-case or session controller represents the control required to manage a use case. This could be used, e.g. in the Take Test UC in MELT. [1]

A lot of people didn’t know this.

d). Suggest whether the use of controllers is appropriate for use in a MELT implementation, briefly explaining why or why not. (2 marks)

My answer would be: no controller is justified in MELT because the application is not complex enough to justify it [1]. Instead, the UI should be designed in a simple, modular way, so that there is a straightforward mapping to the core classes. [1] However, most of them did have controllers (possibly because it was one of the few SE concepts they knew about before starting the course) so coherent arguments in the other direction will be accepted.

Answers were mixed, but this was more about their MELT implementation than the exam question.

e). Explain the notion of a Factory, and how wise use can have advantages in terms of GRASP principles. (4 marks)

Note: factories have been briefly introduced in the course, and will be covered in more detail in the patterns course.
A factory is a pure fabrication whose sole purpose is to create objects of other classes. [1] (Although PF is a GRASP principle, this is not in itself an advantage in GRASP terms). It increases Cohesion by removing complex creation logic from the using classes [1]. Likewise reduces Coupling by simplifying the classes which use it. [1] And (this is the one they’ll probably miss) it provides an Indirection, avoiding explicit constructor calls - the “weak link in encapsulation” - in the using classes. [1]

Most people got at least the general idea, although usually not enough for full marks.

f). Under what circumstances is it appropriate to sacrifice adherence to GRASP principles for advantages in performance (e.g. shorter runtime or lower space usage). (2 marks)

(This is not explicitly taught in the course, e.g. they’ve not come across the Flyweight design pattern yet, so requires thought.)

Very rarely. [1] Modern computers as so fast that minor inefficiencies are irrelevant. Only when there is a clear problem in terms of time or space, should the design be muddied for this purpose. [1]

Answers to this were odd. One common one was for application too simple to need it, but this doesn’t make sense as if simple code is included in larger code, you need GRASP- and it doesn’t really cost anything.

Question 5

a). What is System Testing, as defined in the course? (1 mark)

Testing of the system in the context in which is will be deployed. A common error is to say that it is testing of the whole system (while integration testing is testing of subsystems). These terms are used differently by different people, but the distinction between the development environment and the deployment environment is critical.

This was generally answered correctly.

b). Give four examples of things which would need to be done in system testing for MELT, including at least two which don’t directly involve testing the code. (4 marks)

e.g. verify that the way the software is used is in accordance with the procedures used by invigilators, and what modifications are required to those procedures. [1] Establish training procedures for staff who will be setting and marking tests. [1] Verify that the user interfaces are suitable for all users, not just those who have been involved in the development. [1] Verify that IT personnel have all the information they need to run the server and provide support (e.g. scripts) if necessary. [1]
Marks for any four points which meet the spec of the question.

*People generally came up with enough valid points to get the marks, although not the specific points above.*

c). What happens if, in the course of system testing, new requirements emerge, and what does this imply about the timing of system testing? (2 marks)

You should treat them like any other requirements; prioritise them and consider whether/when to implement them, in consultation with the stakeholders. [1] This of course implies that system testing is not the very last thing done in the project, which indeed it must not be. [1]

A lot of people assumed ST would have to be at the end, and therefore couldn’t answer this bit sensibly

d). Embury’s Law states “This software has bugs in, we just don’t know what they are yet”, where “this software” is almost any system of any size. Explain why this is true, and give one possible counterexample. (4 marks)

When we start testing, we are dealing with a large number of bugs in a huge search space (floating point division is a classic example of this) [1] While the vast majority of bugs may be found by rigorous testing, some will remain in a large system, because they are too expensive to fine. [1] Most large systems are concurrent/reactive, which makes testing much, much harder. [1]

The possible counterexample I’m thinking of is FBW, e.g. in the A320. Although clearly buggy when first introduced, and many Computer Scientists refused to fly in it, a FBW system gives continuous testing for free via redundancy. As many bugs will have been found over the 30 years the aircraft has been in service, so it’s possible that the system really is bug free now. [1] (Certainly nowadays it’s extremely safe).

*Most people got the right general idea, but I don’t think anybody came up with a sensible counter example (trivial systems don’t count)*

e). State four things that can be done in a project to minimise the impact of Embury’s Law, beyond putting a lot of effort into testing. (4 marks)

Focus on mission-critical aspects of the software (bugs in minor aspects of the code are much less important). [1] Document known bugs. [1] Traditional approach: have a separate testing team, who’s job is to be as nasty to the code as possible. [1] Agile approach; write the tests before the code. [1] (The last two are often combined in real mission-critical projects.)
This was generally answered badly (it requires a combination of bookwork and common sense)

f). Estimate how many bugs there are left in your MELT application. You should combine the estimates of but density given in the lectures, with you practical experience of the project, to come up with a realistic estimate. (5 marks)

Note: I ask a variant of this question every year, and the estimates are almost always too high. If the system has hundreds of bugs in, it just won’t do anything. The traditional estimate for code which just compiles is 3-5 per hundred lines. Let’s take 5, and assume their code is 5000 lines long to give us an upperbound of 250 initially. Routine testing, which they will have more-or-less done, gives us a factor of 10, so 25. Rigorous testing, which they will have only partly done due to the tight timescale of the project, in theory gives up to another factor of 10, giving 2.5.

In practice given their inexperience and limited time, it’s highly unlikely that they will have got anywhere near that level. 1-10 per thousand lines of code (5-50 for this example) is more realistic. Marks for a sensible calculation leading to a sensible result. In particular I will be more sympathetic to claims the code is probably bug-free (unlikely but just possible) than to claims that is has several hundred bugs (when nothing would work and it wouldn’t have been possible to give a demo in the lab).

This on the other hand was answered much better than previous years. Most estimates were in the range 10-20, which is much lower, and more realistic, then previous years’ answers.