COMP33411 feedback 2015

Answer question 1 and two other questions. For full marks your answers should be concise as well as accurate. Calculators not allowed.

Questions in normal font, original mark scheme in bold, additional comments in bold italic.

This feedback is based on the original marking scheme.

Overall, with very few exceptions, answers were very good – probably the best I’ve had for an exam on this course.

Question 1

This question is COMPULSORY

a). “Design patterns are a tool for communication”. Explain this statement, using an example. Your explanation should state who is doing the communication and what is being communicated. (4 marks)

Who = developers. What = best practices in software design. Experienced developers who know the patterns can just use the names. Experienced developers can communicate with less experienced ones by explaining the patterns, e.g. with UML diagrams. Full marks for any reasonable example e.g. “do you think these states are interesting enough to represent them as classes using the State pattern?”

A common error is to answer in terms of communicating objects rather than communicating people, but nobody did that this year. The main way people lost marks was by simply not saying four distinct things.

b). Constructo Ltd. make custom toys. They have a range of basic parts (e.g. wheels, cogs, bricks etc.), which are made up into a wide range of different toys, often containing common sub-parts (e.g. a wheel-axle-gearbox combination is common to a number of toy cars). Toys are shipped with just the required parts and assembly instructions, but first each toy much be assembled at the factory to ensure it all fits together nicely. Give a UML diagram which shows how the Composite design pattern can be used to represent these toys. Note: you do not need to represent the different kinds of atomic parts. (4 marks)

Something like the following was all that was required. A few people lost marks for abuse of notation (annotations on inheritance arrows, black diamond, otherwise those knew the composite pattern got full marks and those that didn’t (a minority) didn’t.
c). Briefly explain the advantages of using the Composite pattern rather than having a simple list of parts in the scenario of part b. (2 marks)

It means you can represent composite parts (like the wheel-axle-gearbox combination mentioned in the question) in much the same way as atomic parts (like wheels) [1] and you can do that at any level [1]. So for instance, you could have two toys which were the same apart from a few details, and all the common parts could be represented as one composite.

Some answers to this were a bit muddled, but they were mostly ok.

d). Briefly explain the advantages of using the Composite pattern rather than having a simple list in a different scenario than that of part b. (2 marks)

The obvious example is the ABC software, where we have composite Questions. [1]. That means we can have sub-questions nested to any depth, like a traditional exam paper [1] (and unlike primitive CAA systems like that in Blackboard).

Answers to this were variable, with some very muddled, but mainly ok.
e). Constructo Ltd is a large company and so it any one time it is in the process of shipping many toys, by a variety of forms of transport (e.g. vans, lorries, containers). It needs to keep track of all of these, so, for instance, company managers can see what’s happening and optimise their schedules, and customers can check the status of their orders (as happens with, e.g. Amazon). Draw a UML diagram which shows how the Observer pattern can be used here. (5 marks)

There are many possible variants which will be accepted within reason. Incorrect UML notation will be penalised. And was, although most people knew both the pattern and the notation.

f). Observer is often represented as a two-step process, but actually there are three steps involved. Briefly explain what these three steps are, in the context of the example of part e). (3 marks)

Observer is often called publish-subscribe, but this is misleading because the actual process is:

1. Subscribe – the observables, forms of transport in this case, are registered with the observers. [1]
2. Publish – the observables publish their events to the observers. [1]
3. 3 Act – the observers do whatever is necessary as a result of receiving a notification. [1] In this case, they might update a database with the current position of a transporter, and the toys it is carrying.

People either knew this or they didn’t – about 50-50. There was no evidence that anybody figured it out at the time, although that should be possible,
Question 2

Most people did this question which turned out to be embarrassingly easy.

a). A slogan for agile software development is “embrace change”. One kind of change is requirements change caused by stakeholder feedback. What other kinds of change can happen (to requirements or otherwise) during a substantial software project? (4 marks)

- e.g. A change of direction of the project due to institutional politics or funding.
- Changes of personnel in the development team. Need to conform to new legislation. (e.g. accessibility).
- Technical difficulties causing a scaling down of the project/opportunities causing a scaling up. Marks for any 4 sensible, distinct points.

There were many good answers to this bit.

b). Reminder: Simon Smartsuite is the University’s head of IT, who, while fundamentally on your side, has some very old-fashioned ideas about software process – he still believes in Waterfall!

Simon has “got the right result” with the University’s Senior management team, so the project to build CAA software for the University is on. You decide that it’s time to educate him about the UP, and that it’s hopeless trying to do this via email, so instead you take him out to dinner to thank him for his work with the SMT and to try to explain the advantages of the UP. Give six points you would make about the agile UP, without ever naming the phases of the UP (6 marks)

Note: Simon Smartsuite is a fictional character who the students met in test 1. There, many students parroted the phases of the UP, which in real life would be totally unhelpful. Notably, the students who had taken a year out in industry did far better than those who had not.

Marks for any 6 relevant points, eg.
- We’re applying the UP in an agile way, which means we focus on producing software, and only produce documentation when it’s really necessary.
- We will work in iterations of 4 weeks, which means every four weeks we will have something to demonstrate, from an early stage. (Could also explain timeboxing at this point).
- We will work with real users (NOT the SMT!) from an early stage, to ensure we produce the right product
- We will have working software, which can be trialled with real students, far sooner than the three year timescale of the project.
- Then we will be able to extend from a solid base, implementing new question types etc.
- At any stage in the project, we will be in a position to cope with new, or changing, requirements (not possible with Waterfall)
I marked this bit strictly, not allowing parts which made the same point in different ways etc. The marks were still high.

Note: as is now traditional, the following part of the question is based on the latest real-life University centralisation scheme. Last year was centralised timetabling, and the risks in the two cases are very similar (Unlike last year, I have exaggerated this one a bit)

c). The University of Mancunia decides to implement a University-wide web information system, to go on line for the start of the next academic year, in 9 months time. This will contain a large amount of information, including the web page of every member of academic staff, and a great deal of information for students.

Currently, web pages are held in various different formats in different schools, and over 200 academics and admin staff are involved. Under the new system everybody will use the same software and the information will be stored in a central database which will form the “single point of truth” for all web information in the University.

What will be the main risks associated with this development, as perceived by the following groups of stakeholders?

Note: in the following, marks will be awarded for any sensible concerns not given in the marking scheme below

i). Students (2 marks)

That the information they need may be wrong [1]; that it’s a huge amount of information, and it may be difficult to find what they want. [1]

ii). IT support staff (1 mark)

That the system may require extra work, e.g. having to manage a new, dedicated server. That they might get blamed if something goes wrong.

iii). University management (1 mark)

That there will be damage to the reputation of the University if it goes wrong (e.g. a story in a local newspaper, as happened with the timetabling system at at Salford, or worse).

iv). Admin staff responsible for maintaining web pages (hint: consider both short-term and long-term risks from their perspective). (2 marks)

That they will not understand the new system well enough, and hence screw up; in the short term that the system will be more difficult to use than what they’re used
to, and hence increase workload. [1] In the long term, that the system will be more efficient than previous methods, and therefore require less admin staff to run it, hence job losses. [1]

I was pleasantly surprised how many people got the last point. In general, People made different assumptions about the roles of IT support staff and admin staff, and I allowed answers based on these where reasonable.

d). Of the four groups of stakeholders mentioned above, state for each how important it would be to have a representative interacting frequently with the development team, briefly saying why in each case. (4 marks).

Students - maybe. Addressing their concerns about being able to find things would be important, but this would not necessarily involve frequent interaction.

This can be argued either way, but students ought to be involved to some extent (unlike the way it was actually done here!)

Uni management – no, keep out of your way as much as possible!
Admin staff - yes. As well as getting feedback from them it will be essential to ensure that they have adequate documentation and training.
IT support - possibly, depending on how significant the IT infrastructure changes will be.

Different answers to the above were accepted provided they were justified.

Question 3

a). It’s often said that design patterns are a tool for communication. Explain with an example not given in the course, why the same is true for GRASP principles. (4 marks)

Like design patterns they have names (High cohesion etc.) that can be used as shorthand by experienced designers. [1] They can be used to discuss design tradeoffs (e.g. “by Expert, maybe we should move this operation to that class”). [1] Since they represent best practice in OOSD, they are also good for teaching inexperienced developers, e.g. “that class has poor cohesion, it’s representing two different things, you need to separate them out.” [1] Any sensible example (e.g. from their project) [1].

Most people answered this very well.

b). The Information Expert and Creator principles have the common property that they can both be used “in reverse”. Briefly explain each of these principles, and how they are used in reverse. (4 marks)
Note: I’ve mentioned this for I.E. and Creator is a special case of I.E.

Information expert – put an operation in the class that has the information to carry out that operation. [1] In reverse, put the information necessary to carry out an operation in the class where that operation logically belongs. [1]

Creator – put the creation of instances of a class B in the class A that has the initialisation data for B. [1]. (there are a number of other cases mentioned by Larman, but I think this one is the most straightforward). Converse – put the initialisation data for B in the class A which is most naturally its creator. [1]

Most people were fine with Creator and I.E., but a lot of people were thrown by the reverse bit.

c). “Protected Variations is actually an antipattern”. Explain this statement, and explain why it might actually be true in some circumstances. (4 marks)

An antipattern is a design pattern (or in this case a principle) which actually makes designs worse [1] (this accusation has been made about Visitor for example). With PV there is a tradeoff - we can’s predict what will vary, so we need to be wary of spending a lot of effort on something which won’t happen [1], and of creating over-general designs that are hard to use in the simple case [1] (like a lot of the Java libraries). Agile design principles, for example, discourage designing for situations which may not occur. [1]

Some answered this very well, others were confused.

d). Explain how polymorphism often gives us Protected Variations in a way which is actually useful. (2 marks)

Polymorphism (inheritance), if used correctly, gives us PV in a useful way, because we can add subclasses [1], or change the functionality of existing subclasses [1], without affecting anything else.

This was generally answered well.

e). In the early days of C++, inheritance, and in particular multiple inheritance, was often overused. It was common for a class to have multiple superclasses, with no is-a-kind-of relationship between superclasses and subclasses. Explain, in terms of GRASP principles, why such a design is very bad. (4 marks)

Cohesion: a class with multiple unrelated superclasses will be highly uncohesive. [1] Coupling: it will also be highly coupled, to all its superclasses, for starters. [1] Polymorphism – this principle says use inheritance to improve cohesion and coupling, so such a design clearly violates it. [1] Protected variations – changing such code will have complex and have unpredictable results, so clearly it is also very bad in this respect too. [1]
And this.

f). The Java designers took a very different approach, with only single inheritance, with the exception of interfaces, which have very limited functionality. Common “mixin” cases were dealt with by having some functionality (e.g. synchronization) built in. State whether in your 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 took the view (which they know I take) above. A few people argued in the opposite direction, which would have been fine if they’d come up with an actual concrete example, but none did.

Question 4

a). There are three design patterns which specifically address the problem of representing complex algorithms in OO. Briefly explain how the problem arises, and how each of these patterns can help. (4 marks)

This arises because whereas in a procedural program an algorithm is normally localised in one place (e.g. a set of procedures within a module) in OO it is often distributed among several different classes, making it hard to understand, debug etc. [1] Strategy, by representing algorithms as objects, not only localises an algorithm in one class, but also allows us to switch between them at runtime. [1] Template Method gives us a template for an algorithm which makes it easier to break it down into chunks. [1] Visitor gathers a distributed algorithm in one place, by inverting the normal OO structure.[1]

A few people got the wrong patterns, but most got the correct three and gave good summaries of them.

b). Briefly explain, with an example, why there are many cases where the problem mentioned in part a) simply does not arise. (2 marks)

Often it’s absolutely fine for an algorithm to be distributed across the classes because each does its own part with little interaction between classes [1]. A good
example is toString() where each class contains the code to yield a String representation of itself. [1] (or any other valid example)

Although most people didn’t use toString() as their example, most answers were correct.

c). Explain the idea behind the Flyweight pattern, and why it is different from many design patterns. (3 marks)

Flyweight is a (space) optimisation which significantly increases the complexity of the design – most patterns do the opposite [1]. The idea is that when we have many small objects (characters in a document is the classic example) we can replace them with a smaller number with sharing. [1]. In order to do that we need to take the intrinsic state of an object and make it extrinsic (held separately) [1]

This was generally well answered.

d). At the time the GoF book was written, Flyweight was a commonly used pattern. Nowadays, it is much less often used – explain why. (3 marks)

Memories have got much, much bigger! [1] Flyweight introduces a great deal of complexity into a design which is often not needed. [1] Programmers generally (particularly in Agile shops) are discouraged from making optimisations unless they are clearly shown to be necessary, which in this case they often won’t be. [1]

A surprising number of people didn’t state the first point, which is of course the crucial one. The last point is also worth noting.

e). The GoF book lists a number of conditions which have to apply for the Flyweight pattern to be applicable. One is “The application doesn’t depend on object identity”. Why would this be a problem with the use of Flyweights? (2 marks)

Note: this is non-obvious and I’ve deliberately not told them the answer. Because objects created from a flyweight factory are reused, they are often actually identical. [1]. But this can’t be guaranteed – it depends on the details of the factory and may change over time.[1]

Answers to this were mixed, to say the least!

f). Suppose you have an engineering application which creates many parts. Sometimes there is no problem with doing it the simple way, but at other times there are so many parts that you are forced to use Flyweight. You may not know which situation you are in until runtime. Draw a UML diagram which shows how the Strategy pattern can help in this situation. (4 marks)
Marks deducted for incorrect notation, or clutter not justified by the question.

Most answers to this were fine. A few didn’t make any sort of sense.

g). Why is switching from one strategy to another more difficult in this case than the chess playing example given in the lectures? (2 marks)

Because in that case the strategies were all applied to the same data structure, representing a chess position, so switching between strategies was easy. [1] In this case, we need to convert from one data structure to the other before we can switch strategies. [1].

Again, answer to this were very mixed.

Question 5

This was the least popular question but still had 30 people do it, which I thought was a good result.

a). Explain, using an example other than that in part b the problem which Aspect-Oriented Programming is designed to solve. (4 marks)

The problem of cross-cutting concerns - ones which cut across the natural class structure and are hard to localise within one class. [1] For example logging is a requirement for many operations in ABC. [1] Writing code of the form doIn();
logIt() is uncohesive and couples many classes to the logging API. [1] This could be mitigated in AspectJ by localising the logging functionality in an aspect. [1]

Although most people used different examples, most answers to this were good.

b). Briefly explain what the two main constructs within an aspect in AspectJ are, giving an example of how they could be used to deal with the need to lock certain objects before they are used in a multithreaded application. Accurate AspectJ syntax is not required provided the meaning is clear. (6 marks)

A pointcut marks a well-defined point in the execution of a program. [1] To solve the locking problem we could have a marker interface Lockable inherited by all classes whose instances require locking and have a pointcut something like:

pointcut lockedOperation execution(*.*) && this (Lockable) (i.e. any operation on a Lockable). [2]

Advice is code to be executed when pointcut is reached. [1] In this case we need to lock before the call, and unlock after:

before() lockedOperation(): this.lock(…)

after() lockedOperation(): this.unlock(…) [2]

Answers to this were generally very good. The only thing I cared about (within reason) syntax-wise was that there have to be some wildcards in the pointcut, as it will occur in many places.

c). The use of AOP has been advocated in a large number of different situations. For each of the following, state how plausible the claim is that AOP improves the design, briefly explaining why.

i). In the implementation of CAA software (2 marks)

Yes, because it turns out that CAA has many of the same requirements as the canonical example of banking [1] e.g. locking, concurrency control [1].

Most people got this.

ii). In the use of Design by Contract. (2 marks)

BULLS**T!! (or words to that effect) [1]. This it the “wild-eyed zealot” school of AOP at its most extreme. Since contracts are part of the documentation for the code, they should be with the code (as in the original DbC language, Eiffel). [1] You don’t even end up writing less code by hiding them in aspects, because by and large every class has different contracts.
Quite a lot of people got this wrong, probably because of a lack of understanding of DbC.

iii). In the implementation of design patterns. (2 marks)

This may be true of a small number of patterns (e.g. Observer) but nowhere near the 17/23 claimed by AOP enthusiasts. [1] (This was the conclusion of an excellent MSc project). The reasons vary, but for example in the case where the pattern code is a pure fabrication putting it in an aspect achieves nothing except making the code harder to follow. [1]

This was discussed at some length in the course, and most people who did this question had the right idea.

d). State the advantages and disadvantages of AOP which should be taken into account when deciding whether or not to use it on a project. (4 marks)

The advantage is obviously being able to deal with CCCs in a modular way [1] In some situations this may outweigh the disadvantages, which include:
3 of: All developers, present and future, need to be familiar with it; it constrains what languages and tools we can use; overuse can lead to poor design; to fully understand any code which uses aspects, you need to look in two different places. (Answers which stress the advantages more are fine, so long as they make four separate points.)

Many answers to this were not necessarily what I would have first thought of, but good enough for the marks.