Comments

Please see the attached.
Examination Performance Feedback to COMP61532 (2014 – 2015)

Section A
Answer ALL Questions

In general we both found the answers depressingly poor – probably the worst we’ve seen on an MSc exam.

a) What is a software process? Why do we need software processes? (2 marks)

A software process is a structured set of activities for specifying, designing, implementing and testing a software system [1m]

Successful software engineering projects show: (1) good software development processes lead to good software and (2) good processes reduce risk [1m]

This question is a straightforward bookwork question as its answer is provided in the lecture notes.

Only a couple of students answered the questions correctly. Two common errors: (1) the students don’t known what activities are involved in a software process; (2) the students don’t know the benefits of software processes.

b) Explain the four main stages in the incremental development process. (4 marks)

• Decompose the development and delivery into increments with each increment delivering part of the required functionality [1m]
• Prioritise user requirements and include the highest priority requirements in early increments [1m]
• Frozen the requirements once the development of an increment is started [1m]
• Continue to evolve the requirements for later increments [1m]

This is also a bookwork question as its answer is provided in the lecture notes.

Again, only a couple of students answered the questions correctly. A common error is that some students mixed the stages in this process with the stages in the waterfall process.

c) Explain the four main stages of the extreme programming process. (4 marks)

• Planning: Use user stories and acceptance criteria to plan different iterations
• Design: Produce a simple design or prototype [1m]
• Coding: Relies on constant code improvement, user involvement in the development team and pair programming [1m]
• **Test:** continuous testing and user acceptance testing [1m]

This is also a bookwork question as its answer is provided in the lecture notes, but only a handful of students got the answer right. Two common errors: (1) the planning stage is answered as the requirements stage; (2) the design stage is missing.

d) *Concisely* describe what design patterns and GRASP principles are, and how they are related.  

(4 marks)

Design patterns express non-obvious solutions to non-trivial design problems. [1] GRASP principles are lower-level and more general principles for designing good software. [1] Patterns promote GRASP principles of high cohesion and low coupling etc. leading to clean designs which adapt to change. [1] Enhancement of GRASP principles is a good indicator the a pattern is applicable. [1]

Something like the first two points is necessary to answer the question. Other valid points will be accepted for the second part, provided they are concise.

*Likewise, few students got all the relevant points.*

e) *Briefly,* without the use of a diagram, explain how the **State** pattern works.  

(4 marks)

The basic idea is that if something has a number of different states, model those states as objects, with a different class for each state. [1] There is an abstract super-class, which contains attributes and operations which are independent of state. [1] It also has abstract methods, which vary with state and are implemented in the subclasses representing the different states. [1] State transition code may be implemented either within the state classes themselves or separately. [1]

*Overall answers were very mixed. Even those which got the marks were rarely well phrased.*

f) Under what circumstances is it appropriate to use the State pattern and what is the most common alternative?  

(2 marks)

When the differences between states are non-trivial so that the implementations of the operations in the different subclasses are substantially different. [1] If the differences are trivial it’s better to have a single class and implement differences between states with enums and switch statements etc. [1]

This is intended to be tricky. Nowadays I consider State rather than switch statements more common than it used to be, because the efficiency differences are less relevant.
Ditto.

**Section B**

**Answer ONE question from this section**

Both questions in this section are based on the following scenario.

You are designing a software system which will help in the management of large hierarchical organisations such as armies, governments or universities. The aim is to provide a general-purpose system which can be customised for different organisations. The system will keep track of the personnel in an organisation, and also the material resources it requires.

All such organisations consist of sections, which have subsections, and so on. The software cannot make any assumptions about how many sections an organisation has or how many levels there are in the organisation. Also, different organisations have different names for their sections (e.g. “regiment”, “brigade” in an army or “faculty”, “school” in a university).

We can, however, assume that 1. Each section has a person who leads it, with a job title and a name, and 2. Every organisation has a unit section, which is not divided further, but which normally has more than one member of staff (such as a platoon in the army).

**Question B1**

a). Draw a UML class diagram which shows how the **Composite** design pattern could be applied to model organisations as described above. (5 marks)
The key point is the dual relationship between CompositeSection and Section. For full marks the diagram should look more or less exactly as shown above. An acceptable variant is to have an extra class representing the personnel in the unit section; also this could be called AtomicSection. Leaving the multiplicities out could be allowed but if they are there, they must be correct. For solutions with the right overall shape, deduct marks for:

- Leaving out the attributes mentioned in the description, or adding extra ones which aren’t there.
- Abuse of UML notation, in particular the diamond for “contains” and the triangle for inheritance.

A solution which has CompositeSection containing unit sections (as opposed to section) should get max 3. A solution which is the wrong shape altogether should get max 2.

The answers to this were disappointing. All the above errors were made.

b). Explain how the collective skillset of all the people in an organisation could be calculated based on this design. Hint: this requires set operations, rather than integer ones. (3 marks)

The calculateSkillSet() method would be declared abstract in the abstract Section class. [1] A unit section would implement this by taking the set union of all the people in the section.
A composite section would do so by recursively taking the set union of the skill sets of its subsections. [1]

Note the second point is different from other examples of Composite they’ve seen where the atomic operations made use of single stored values, so they must get that part right for full marks.

*Again, I struggled to find ways to give marks for this.*

c). Large organisations generally have large numbers of items of equipment, for example an army will have a very large number of bullets. Briefly explain what a factory is and state two advantages of using one of using one in this situation. (3 marks)

A factory is a Pure Fabrication whole sole responsibility is to create objects [1], in this case bullets. It decouples code to create software objects corresponding to the real bullets from code using them. [1] Many bullets are identical and so a software AmmoFactory (unlike a real one) can store a single instance of each kind of bullet and return a reference to it each time an instance is requested. [1]

*Ditto.*

d). Suppose we have a particularly conscientious army which wishes to keep track of every single bullet it possesses. A bullet can be in number of states, e.g. in storage (at some place), allocated (to some unit), or used.  

i). You could use the State pattern to represent the states of bullets, but briefly explain why this is may not be necessary, and what you could do instead. (2 marks)

Because bullets are relatively simple things with relatively simple states, [1] so a single Bullet class (or a small number for different kinds of bullets) with conditional code depending on state would probably suffice. [1]

As above, I probably would use State now, but this part is about considering alternatives.

*Ditto*

ii). Briefly explain the principle behind the Flyweight pattern (2 marks)

We distinguish between the intrinsic state of an object, stored within it, and extrinsic state, stored elsewhere. [1] The idea of the Flyweight pattern is to reduce the amount of intrinsic state, replacing it by extrinsic state, in order to increase sharing and hence reduce the number of objects needed. [1]

*Ditto*
iii). Explain how Flyweight could be applied to this particular situation, in a way that minimises the amount of storage required to represent bullets. (3 marks)

We need to find places to put the state information which would otherwise be held within each bullet object. [1] There will need to be several different such places, representing ammunition depots, operational units etc. [1] If all of the state is made extrinsic we don’t even need references to bullet objects at all, just counts of the numbers of each kind of bullet in each place in each state. [1]

Ditto

iv). How is the Flyweight pattern related to GRASP principles? (2 marks)

Flyweight itself isn’t. It is an optimisation which is actually likely to increase coupling and generally complicate the design so actually goes against GRASP principle [1] It does rely on a factory which is a Pure Fabrication which increases cohesion and reduces coupling etc. [1] (Any sensible variant on the second point is ok, but the first point must be there.

A few people got the right general idea here.

Question B2

a).

i). *Data driven programming* means taking information out of the code and storing it as data which is read by the code. Explain how the **Interpreter** pattern uses, and extends, this idea. (2 marks)

**DDP allows us to change the behaviour of a program without changing the code itself.** [1] The Interpreter pattern extends this by inventing a language for the data which is interpreted within the code.

*I was stunned how little many people knew about Interpreter.*

ii) Different organisations (as described above) calculate overtime payments in broadly the same way, but details such as the total amount of overtime allowed vary both between organisations and between sections within them. How could the Interpreter pattern be used to enable us to calculate overtime payments without writing many different versions of the overtime calculation algorithm.? (3 marks)

We write a generic, parameterised algorithm for overtime payments. [1] We define a language which describes the parameters which vary between and within organisations. [1] We interpret the data for a particular organisation or section to give a customised version of the generic algorithm. [1]

Ditto.
iii) How is the Interpreter pattern related to GRASP principles? (2 marks)

An interpreter is a Pure Fabrication [1]. Its main role is to protect against variation by wiring less information into the code. [1] (Marks for other valid points, although these are the most obvious ones, to me at least.)

Bizarrely, some answers to this suggested that people knew more about Interpreter than the first two parts suggested.

b).

i) Airlines have aircraft which may be in one of several states, e.g. they may be in the air, on the ground being serviced, in a hanger being maintained etc. State changes may happen at any time (e.g. an aircraft may become unserviceable at short notice) and may need to be known about in multiple places. Explain how the Observer pattern can be used to deal in this situation (4 marks)

The aircraft are the observables and the places where state changes need to be known are the observers. [1] We could have a class hierarchy of each (e.g. for different aircraft types). [1] Observables would register their interest in particular aircraft (e.g. a regional maintenance facility could register with the aircraft in its region. [1] When an aircraft changes state, the software object associated with it cases all observers are notified. [1]

Most people at least had the right general idea here.

ii) Other organisations have similar situations but with different equipment and state changes. For example an ambulance service needs to keep track of its ambulances. Draw a UML class diagram to show how the Observer pattern can be used to track both airplanes and ambulances, in a way that can be extended (by adding extra classes) to other cases. (5 marks)

Guidelines for marking this are similar to those for the Composite example above, although there is rather more scope for variation, in particular any sensible names for the observers are ok. If publish and/or subscribe (or synonyms) are shown as methods they should both be in the Observable class despite normally being shown in opposite directions as above. (See below)

Overall, this was a bit better done than the one in the previous question. The most common problem was adding extra information that isn’t in the question.

iii) The Observer pattern is often described as publish-subscribe, but explain, using examples from question b ii) above, the actual sequence of operations and where they are implemented. (4 marks)
The actual sequence is subscribe-publish-act. [1]
First, an observer (e.g an AirlineObserver) has to subscribe to an observable, but the subscribe() operation itself involves being added to a list in the observable (Airline) and so must be implemented there. [1]
An observable e.g. Airliner then publishes to all its registered observers. [1]
The observer may then act on the information published (although in this example, in most cases they won’t [1] (Although on 9/11 they most definitely did!)

Some people got this right, others didn’t have a clue, and some found incredibly long-winded ways of saying subscribe-publish-act.
Section C
Answer ALL Questions

The questions in this section relate to the design of a domain model for an online bookstore. The design reuses some transaction and accounting patterns you have learned from this course.

a) Use two transaction patterns to model the situation: “The customer has placed two books in her shopping basket.” Draw a UML class diagram to illustrate this model.

(4 marks)

Sample class diagram:

- Show correct classes [1m]
- Show correct relationships [1m]
- Show correct multiplicities [1m]
- Show key attributes [1m]

None of the students received a full mark for this question. Common errors are: (1) missing the Customer class; (2) incorrect using the aggregate relationship; (3) incorrect using multiplicities. Other types of error: A few students used two separate diagrams to represent this model, which is clearly incorrect. About two students represent the Book class into two classes: Book1 and Book2, indicating that they mixed the concept of class with the concept of instance.

b) Name these patterns. Explain how they have been used in your model.

(3 marks)
Two patterns are Line Item and Container-Content [1m].

Line Item represents the relationship between a book to purchase (in the shopping cart) and a book as a product. [1m]
Container-Content represents the relationship between the shopping basket and a book to purchase (in the shopping cart). [1m]

Most students can answer this question.

c) Explain the relationships between the domain classes and the meaning of the multiplicities on the relationships.

(3 marks)

The customer uses (is associated with) a shopping basket  {1}:{1} [1m]
The shopping basket contains 2 books to purchase {1}:{2} [1m]
A book can be purchased by more than one customer and hence can be contained in more than one Book to Purchase  {1}: {1..*} [1m]

N.B. The purpose of this question is to test if the students can use the containment relationship and the multiplicities between objects correctly.

Most students cannot explain the type of relationship and the multiplicity of the relationship.

d) Extend your domain model so that it can include this situation: “The customer pays for the books by using her credit card and the bookstore’s internal accounting system records the sale by creating an accounting entry.” Use two transaction patterns and two accounting patterns to model this situation. Draw a UML class diagram to illustrate this model.

(6 marks)

Sample class diagram:
Show correct classes [2m]
Show correct relationships [1m]
Show correct multiplicities [2m]
Show key attributes [1m]

None of the students has answered this question correctly. Common errors: (1) missing Descriptor class; (2) missing Accounting Entry class; (3) incorrect relationship between Sale and Book to Purchase classes.

e) Name these four patterns. Explain how they have been used in your model.

(4 marks)

These four patterns are: Transaction-Line Item, Transaction-Transaction, Event, and Accounting Entry. [give 1m if this is the only answer. No mark is awarded for this if the student cannot explain how these patterns have been used in their model]

Transaction-Line Item represents the relationship between the sale and the books to purchase. [1m]

Transaction-Transaction represents the relationship between the sale and the credit card payment. [1m]

Event represents the relationship between the sale (event) and the card payment (subject). [1m]
**Accounting Entry** represents the relationship between the sale (event), the accounting entry and the entry description. [1m]

*About half of the student did well with this question. The most common error is that a considerable number of students failed to recognize the Transaction-Line Item pattern. Another common error is no explanation is given what relationships these patterns represent.*

NB: Questions d) and e) are considerably hard than questions a), b), c) in that not only does it require the students to apply more patterns, but more crucially, it also asks the students to combine two types of pattern together. This question tests the students’ understanding of analysis patterns at a deeper level.