COMP23420 exam performance feedback 2014

Questions in plain font, answers in **bold**, additional comments in ***bold italic***.

General comments:

Liping: This year's performance on my questions was generally poor. I believe this has a strong correlation with poor lecture attendance on the course, as all my questions are based on my lecture notes. Common mistakes are:

- ***Questions were partially answered.***
- ***Concepts or approaches were not properly understood.***
- ***Problems with drawing UML Class Diagrams.***

John: Totally agree with the above. My parts of the questions were marked very generously, so the average is ok, but around 10 people failed outright (although may not necessarily have to resit, as the performance over the whole two semesters is what counts.) Really, this was a very easy exam (probably too easy in fact) with a lot of bookwork.

Question 1

a) Given the following rules:

1. "There can be no more than 50 hours driven by any one driver in any one week."
2. "There can be no more than 5 hours driven by any one driver without a break."
3. "A driver may work up to 9 hours in a single day."
4. "A driver may specify up to two resting days for each week in which they will not be available for work."

State which rule(s) are for driver scheduling and which ones are for driver rostering.

(4 marks)

Rules 1 & 4 are for rostering [1m each]; rules 2 & 3 are for scheduling [1m each]

This bookwork was answered mostly ok.

b) Briefly explain why driver scheduling is a hard problem and the methods for solving this problem.

(3 marks)

- **The purpose of driver scheduling is to produce a driver schedule to cover all the bus work in a given bus schedule so that every bus has a driver duty assigned to it at all time. [1m]**
- **The purpose of driver rostering is to produce a driver roster to cover the driver schedule so that every driver duty has a real driver allocated to it. [1m]**
- **Difference: Driver scheduling deals with the duty assignment rather than real driver assignment. Driver rostering combines the driver duties with the rest days to form work plans for drivers. [1m]**

Common mistake 1: most students only partially answered the first two points and omitted the third point. A typical answer from students is: "Driver scheduling is about assigning driver duties to buses and driver rostering is about allocating real drivers to duties."

Common mistake 2: a small number of students partially answered the third point, but omitted the fact that driver rostering needs to combine driver duties with driver rest days to form rosters for drivers.

Given that the students have spent the whole semester 2 on developing a driver rostering system, these mistakes seem to suggest that most of the students have failed to have a clear understanding what driver rostering is for.

c) Explain the principles of data abstraction, information hiding and encapsulation in Object-Oriented Programming.

(3 marks)

Data abstraction: the process of defining classes by focusing on their relevant information and leaving out their irrelevant details [1m].

Encapsulation: used in two senses:

1) **Restricting access to a class's components. [0.5m]**
2) **Biding the data with the methods (or other functions). [0.5m]**

Information hiding: the process of separating the class interface from the class implementation [0.5m]. The purpose is to provide a stable interface which protects the remainder of the program from the implementation (the details that are most likely to change) [0.5m].

This bookwork part was answered mostly ok.

d). Why is it important for a software development team to have physical meetings rather than virtual ones whenever possible?

(2 marks)

Because physical interactions are richer than virtual ones – we can see body language etc. [1] Because in virtual interactions people often hide behind personas which are generally negative and unhelpful. [1]

There were a number of good answers to this, with other valid points which I gave marks for. Some anwers failed to say 2 different things.

e). Briefly explain the two types of coupling defined in the lectures.

(2 marks)

Internal coupling is coupling within a subsystem. [1] External coupling is between subsystems [1] (Not between systems).

The majority of people got this right.

f). For each of the types of coupling you stated in part d, give an example of how you kept it low in your IBMS implementation.

(2 marks)

For external coupling, model-view separation is the obvious one. For internal coupling, it will depend on the details of their implementation.

Quite a number of people got this wrong (often despite giving a correct answer to part e), typically citing examples of external coupling as internal.

g). What is the difference between *integration testing* and *system testing*, as defined in the lectures?

(2 marks)

Integration testing is testing of a system or large subsystem in the development environment. [1] System testing is doing it in the deployment environment. [1] (NOT, as defined in the lectures, testing the whole system rather than a subsystem).

About half of the answers to this were correct, most of the others did the thing marked NOT. There are a number of different uses of these terms in the textbooks – I picked the one that I thought made the most useful distinction.

h). Why, in a conventional approach to testing, is it important for the person who tests the code to be different from the one who wrote it?

(2 marks)

Because the person who writes the code will have (perhaps unconsciously) an attachment to the code, and not test it to fail [1] while another person can be arbitrarily nasty to it. [1]

Mostly answered ok, a lot of answers waffled and sometimes still failed to make two distinct points.

Question 2

The vast majority of students chose this of the two optional questions.

a) What are the three basic views captured by object-oriented programming? Briefly explain how each view is represented in UML diagrams.

(3 marks)

The three basic views [0.5m each]:

- **Structural View: Describing the static structure of the classes in a system and their relationships**
- **Behavioral View: Describing the dynamic, runtime behavior of a system**
- **Functional View: Describing the processes and data flows of a system**

Their corresponding UML diagrams [0.5m each]:

- ***Structural View***
  - **Class Diagram**
  - **Object Diagram**
- ***Behavioural View***
  - **State Diagram**
  - **Interaction Diagram: Sequence Diagram & Collaboration Diagram**
- ***Functional View***
  - **Activity Diagram**

This was mostly answered ok.

b) Explain what two common program organisations are and what object-oriented problems these organisations attempt to address.

(4 marks)

Two common program organisation [1m each]:

- **Software Architectures: Decompose a system into a set of larger organisations (top-down)**
- **Design Patterns: Structure classes into larger organisations (bottom-up)**

These organisations attempt to address the following problems [1m each]:

Classes not rich enough to represent real world applications

- **Too fine-grained and at a too lower level of abstraction**
- **Class relationships are too generic**

There was a general lack of understanding in answering this question as only a couple of students answered the question correctly. In some answers these two methods were "Strategy Pattern" and "Role Pattern", whereas in others, these methods were "layered architecture" and "MVC".

c) Briefly explain the role played by each of the components in the MVC architecture.
(3 marks)

There are 4 components in MVC

- **Model implements application logic & performs the computation [0.5m]**
- **View displays the computation results [0.5m]**
- **Controller coordinates between the Model, the View and the User [1m]**
- **User (which can be another system) initiates the computation through the input command [1m]**

d) Briefly explain the role of GRASP principles in object-oriented software development.
(2 marks)

A good answer is

They provide a set of principles for assigning responsibilities to classes, the most difficult skill in OO software development.

+ one other good point, e.g. Like all patterns they form a language which helps developers to communicate They provide guidance on when to refactor and how.

Most answers to this were ok.

e) Explain how your IBMS implementation was, or could have been, consistent with **five** different GRASP principles, *other* than low coupling.
(5 marks)

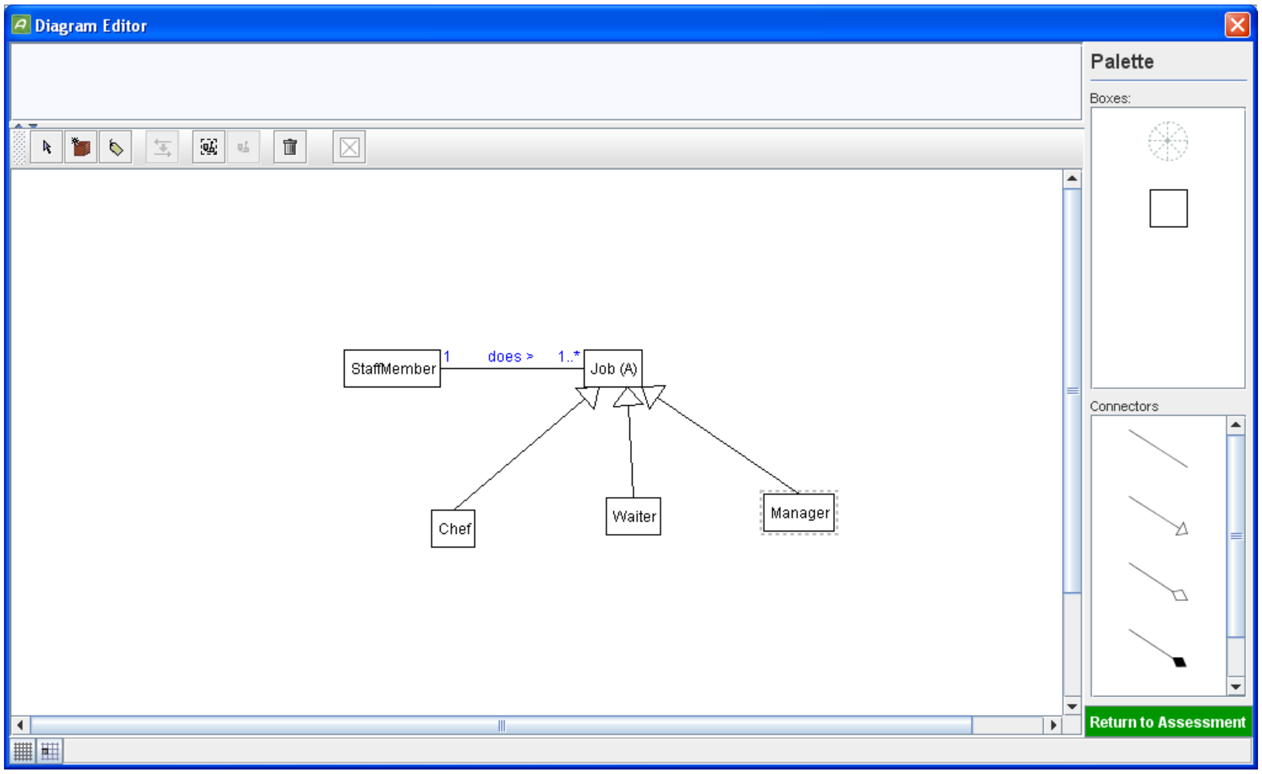
Excluding low coupling leaves High Cohesion, Polymorphism, Protected Variations, Information Expert, Creator, Pure Fabrication and Controller. There isn't much scope for Polymorphism in the problem, and they may not have a controller, but all the others should be there. Marks for relating five different principles to the problem, not just for listing or explaining them.

And also for this. One positive thing was that most people knew the GRASP principles.

f) "A restaurant has different kinds of staff members – waiters, chefs, managers etc. One member of staff may do different jobs at different times, e.g. a manager may act as a waiter."

There are two different ways to model the information above using inheritance – one is correct, the other is not. Draw a UML class diagram which shows the correct way. For full marks you need to use correct UML notation as well as the correct model.

This is of course the roles problem, so the solution is:



No marks for clearly incorrect solutions, for ones with the right shape, marks deducted for incorrect use of UML notation (triangle, diamond) or unnecessary clutter.

While the majority of people had the right general idea, all but a handful lost marks through improper use of UML notation (even after I allowed an open diamond on StaffMember, which is not strictly correct as a staff member doesn't contain a job). For example, Inheritance relationships DO NOT have multiplicities – it makes no sense. This matters – UML is a tool for communication, but only if you use it correctly. I this respect, my first year business school students did better.

Question 3

Note: solution diagrams for parts a) and b) on the marking scheme omitted here.

You are designing a point-of-sale software system for a convenience store. You want to apply some of the business application patterns you have learned from this course to your design.

a) Use a UML class diagram to represent a shopping cart (assuming the shopping cart is not empty). Your diagram should show: (1) the multiplicities of these relationships; (2) essential attributes of these objects and essential operations performed by these objects. Explain the business patterns used in your design.

(5 marks)

The correct class diagram should consist of two business patterns: Item-LineItem and Container-Content (for basket and LineItem). In addition, the diagram should have correct representation of multiplicities, essential attributes and operations.

Common mistakes: (1) incorrect use of multiplicities between Item and LineItem classes; (2) omitted LineItem Class; (3) omitted the containment relationship; (4) omitted attributes and operations.

b) Use a UML class diagram to represent a cashier who is processing the sale and the sale is happening in the store. Your diagram should show: (1) the multiplicities of these relationships; (2) essential attributes of these objects and essential operations performed by these objects. Explain the business patterns used in your design.

(5 marks)

The correct class diagram should consist of two business patterns: Participate-Transaction (cashier-sale); Container-Content (Store-sale). In addition, the diagram should have correct representation of multiplicities, essential attributes and operations.

Common mistakes: (1) incorrect use of multiplicities between Store and Sale classes; (2) omitted cashier or sale class; (3) omitted the containment relationship; (4) omitted attributes and operations.

c). Briefly explain the term “bug density” as used in the lectures, and how it is related to the total size of the code.

(2 marks)

Bug density is the number of bugs per lines of code, e.g. 1 per 100. [1] It increases with total code size, as interactions between subsystems become more complex. [1]

Most people omitted the second point.

d). Suppose the complete IBMS system is around 1.5 million lines of code. It is written mostly in Java by a highly competent team using agile practices such as pair programming and is thoroughly tested. Using the estimates of bug density given in the lectures, approximately how many bugs would you expect to be in the system at the point is deployed? Hint: first calculate the *minimum* there are likely to be.

(3 marks)

The estimate given in the lectures is 1-10 per 100 lines of code, before any testing or debugging, so the lowerbound for this is 1 per 100. [1] After really rigorous testing, we can reduce this by a factor of 1000, giving a lowerbound of 1.5M /100K = 15. [1] Given the description of the development process, we would expect to get quite close to the lowerbound, but it is quite a big system. My guestimate would be 50-200. Anything in the right ballpark will get the mark. [1]

Most people seemed simply to not have read the question properly – this is a thoroughly tested system, and most estimates were unrealistically high.

e). After deployment, would you expect many of the remaining bugs to be found quickly? Briefly state why or why not.

(1 mark)

No, because most or all of the "obvious" bugs will have been found, and those remaining will be in obscure situations, and / or be difficult to reproduce, such as concurrency bugs.

Most people got this right.

f). The complete IBMS system would allow the controller to deal with all sorts of situations such as bus breakdowns, driver illness etc. Since a controller will have been working with the development team throughout, you can be confident that the user interface is satisfactory, and that the required functionality works correctly in the development team's office. State two things you would need to check carefully during system testing of this part of the system.

(2 marks)

That it works correctly in the controllers office! [1]. That the required documentation, procedures and training are in place to enable other controllers to use the system. [1]

There were all sorts of answers to this, mostly missing the point (or again, not reading the question properly.

g). The complete IBMS system would allow passengers to access timetable, route planning and real-time information on a range of mobile devices. State two things you would need to check carefully during system testing of this part of the system.

(2 marks)

That the UI is satisfactory for variety of passengers with different ways of using the system [1] across the full range of devices supported. [1]

Answers generally got the second point, but not the first.