School of Computer Science

EXAMINATION PAPER: COMP23420

MONTH AND YEAR: Feb 2012

SET BY: Liping Zhao, John Sargeant

QUESTION NUMBER: A (Bookwork + analysis).

LEARNING OUTCOMES ASSESSED: Understand the basic concepts of software architecture and high-level design constructs, testing concepts, basic GRASP design principles.

MARKING SCHEME

a) What is the name of the software architecture used in your IBMS project? Provide a general description of its constituent parts and then relate these parts to the specific components in your project.

(2 marks)

It is a layered architecture consisting of a UI Layer, a Domain Layer and a Database Layer (1 mark).

In IBMS: a rostering UI, a design object model for rostering and Java wrapper classes (1 mark).

Minor variations on the above were accepted, for instance although the wrapper classes, along with the database tables, are strictly speaking part of the data management layer, answers which put them in a separate layer were ok.

A number of people claimed to have used MVC. This was accepted iff there was a clearly described controller component, separate from both the UI and the application logic. In most instances this was not the case.

Although the majority of answers were correct, several confused software architecture and physical architecture (one actually said "We used client-server physical architecture") and some people even talked about development methodology rather than architecture. There's no excuse for failing to read a straightforward question like this correctly.

b) What are the two main roles played by a physical architecture?

(2 marks)

1 mark each role:

1. **It determines how the software components of an application system are distributed to the hardware and network environment of the system**
- **It also determines how the non-functional requirements (NFRs) of the application system (e.g. operational, security and human-oriented requirements) are supported by both hardware and software components**

Although there are different answers possible to this question, the two points above were strongly emphasized in the lecture, and only these were accepted. Many people got both points correct, although the second appeared less often than the first.

c) In the context of object-oriented design, what are design abstraction and refinement?

(2 marks)

1. **Abstraction: if classes share attributes and methods, "factor them out" and create a superclass to represent their generalization.**
2. **Refinement: Identifying additional subclasses (of a higher level class) which have non-trivial differences between them.**

These words could mean a wide variety of things (and the space of possibilities was thoroughly explored in the answers) but these specific definitions were given in the lecture notes and emphasized in the lecture.

d) Concisely and critically describe the purpose of partitions and layers in software organization.

(2 marks)

- **Purpose of partitions (1 mark): Creating subsystems, i.e. <u>collections</u> (usually sets) of related (modular) classes. In object-oriented designs/implementations the "pattern of activity", i.e. messages sent between objects, provides one basis for partitioning system into (smaller-scale) subsystems (where subsystem = *partition*)**
- **Purpose of layers (1 mark): Layer enables <u>useful</u> separation of (related) concerns, e.g. separating application logic from user-interface considerations. Layer constrains the kinds of class that exist "on that layer"**

Many variations of the above were accepted. Answers with examples were good. Many answers were too vague to get full marks, in particular, e.g. not recognizing that partitions and layers are different in kind as well as scale. Phrases like "separation of system concerns" were not accepted without some evidence that their meaning was understood.

e) Give two non-functional requirements that you have learned from this course and briefly explain their implications to architectural design.

(2 marks)

One mark for choosing any two from the following:

- **Operational NFRs**
  - **System Integration**
  - **Portability**
  - **Maintainability**
- **Performance NFRs**
  - **Speed**
  - **capacity**
  - **Availability and Reliability**
- **Security NFRs**
  - **System value estimates**
  - **Access control**
  - **Encryption and authentication**
  - **Version control**

One mark for answering the implication of each NFR to architectural design according to the following mappings.
(NB: students are not required to draw this table. I provided the table here for marking purposes)

Requirements	Server-Based	Client-Based	Thin Client–Server	Thick Client–Server
Operational Requirements				
System Integration Requirements	✓		✓	✓
Portability Requirements			✓	
Maintainability Requirements	✓		✓	
Performance Requirements				
Speed Requirements			✓	✓
Capacity Requirements			✓	✓
Availability/Reliability Requirements	✓		✓	✓
Security Requirements				
High System Value	✓		✓	
Access Control Requirements	✓			
Encryption/Authentication Requirements			✓	✓
Virus Control Requirements	✓			
Cultural/Political Requirements				
Multilingual Requirements			✓	
Customization Requirements			✓	
Making Unstated Norms Explicit			✓	
Legal Requirements	✓		✓	✓

FIGURE 13-16 Nonfunctional Requirements and Their Implications for Architecture Design

It rapidly became apparent when marking this part that there is an ambiguity in the question. While the intention was to relate NFRs to physical architecture design, the word "physical" is missing from the question, and hence some people interpreted it as referring to software architecture design. Such answers were accepted provided they ware specific about how the software architecture design was affected by the NFRs - this is a significantly harder question to answer than for physical architecute!

A number of answers were far from brief, and wolfed on about NFRs without relating them to any kind of architecture design, as if the question had been "tell me about as many NFRs as you can think of.

Sensible suggestions were accepted whether or not they corresponded to the list and table above. A number of people answered based on their experience with the IBMS, which was fine iff the answers related NFRs to architecture design.

f). Briefly explain the GRASP principles of **high cohesion** and **low coupling**. Your answer should differentiate between the two kinds of coupling explained in the lectures. (3 marks)

High cohesion: a class should represent a single well-defined entity. [1] Low coupling: dependencies between classes should be minimised - not just the number of dependences but the nature of them, e.g. stability. [1] Internal coupling is coupling within a subsystem; external coupling is coupling between a subsystem and other code. [1]

Since this is bookwork it was marked strictly. Since the questions said "briefly explain" a few answers lost a mark purely on length. Some people said that external coupling is between the system and external entities such as databases. While this is a plausible definition of external coupling it is not the one given in the lectures. The distinction is important because GRASP principles are about low-level class design (for example as in part g below), as opposed to the higher-level design issues discussed in Liping's later lectures. Another common error was to say that high cohesion meant that design classes should correspond directly to domain classes. It's common to have to split up domain classes, or introduce pure fabrications, in order to keep high cohesion.

g). Explain in terms of cohesion and coupling whether in the IBMS scenario it would have been a good idea to have a single class which integrates the roster and the driver timetables it produces. (2 marks)

No because: Such a class would not be cohesive, as a driver timetable is a well-defined entity in itself and should be a separate class. [1] It would have high *external* coupling, as the rest of the system depends on the roster. [1] (A common error is to assume that a single class has no coupling, but this only considers internal coupling.) A precise answer is required for full marks here.

The common error was indeed commonly made. Very few answers explicitly mentioned external coupling, and I accepted answers which sad essentially the same thing without using the term. A lot of people worried about coupling within the combined class, but this is not a big deal - it could be written in such a way that the code was not much different to the code for two separate classes. Also, a number of answers, while recognising that it would not be a good idea, did not explain this coherently in terms of cohesion and coupling.

A significant minority of people actually thought it would have been a good design.

h). What is the key idea of JUnit testing?. (1 mark)

You write unit tests, separately from the code being tested, in the form of assertions.

With hindsight, this was not a very well phrased question, and a wide variety of answers were accepted. However, just "unit testing" (without at least saying clearly what unit testing is) was not enough.

One way or another, the vast majority of answers got the mark. Those that didn't were mostlyy too vague - taking about testing in a very general way rather than unit testing.

i). State two advantages of JUnit testing. (2 marks)

Any two of: it allows you to automate tests; by separating the tests from the code it makes it easier for the coder and tester to work independently; it can give you a good level of confidence of individual units (classes) before you try to integrate them.

A number of people related JUnit testing to TDD and agile development generally, which was good. A number of answers mentioned specific features of JUnit such as annotations and parameterisation. These were ok provided the advantages of these were stated. General comments such as "simple and easy" were not accepted as there are many more specific points which can be made.

j). State two disadvantages, or limitations, of JUnit testing. (2 marks)

Any two of: It only applies to unit testing, so it doesn't directly help with the hard bugs which can only be found in integration or system testing; it only apples to tests which can be stated in terms of assertions, so it doesn't help with GUIs for example; having an assertion fail does not necessarily mean there is a bug in the class being tested since classes depend on other classes.

A number of answers were not accepted because they assumed particular ways of going about JUnit testing and would not apply if things were done differently. For example although JUnit is often used for back box testing, it doesn't have to be, if you have access to the code you can do white box too. Similarly JUnit testing can be done TDD-style or conventionally, or a combination of both. Also, there were a lot of general comments which would apply to almost any kind of testing - comments like "time consuming and expensive" fall into this category. Similarly, if functionality of the code changes the tests must also change however you do testing. A number of people thought that JUnit can only be used with and IDE (oreven a particular IDE). I gave an example of running from the command line.

One good point which a number of people made is that since the test classes are separate from the application classes, you can only test the public interface, not e.g. private helper methods (even if you are doing white-box testing). I reluctantly gave a mark for saying it only works with Java!

School of Computer Science

EXAMINATION PAPER: COMP23420

MONTH AND YEAR: Feb 2012

SET BY: Liping Zhao

QUESTION NUMBER: B1 (Problem-solving + Analysis).

LEARNING OUTCOMES ASSESSED: Understand MVC, its components, their interactions and their mappings onto the client-server architecture; understand the similarity and difference between MVC and layered architecture.

MARKING SCHEME — 20 MARKS

B1. This question consists of the following two parts:

a). Clearly and concisely describe:

i The components of the MVC software architecture. (3 marks)

A Model implements application logic. Views and Controllers manage user interfaces, where a Controller deals with user input and a View deals with presentation /output.

The question clearly says "describe" so merely saying MVC stands for model-view-controller got no marks. It should be obvious that for 3 marks (6 minutes) you should expect to do more than just write three words. On the other hand, many people effectively included the answer to the next part in this part, resulting in answers which may have been clear but not necessarily concise.

Several people confused models and controllers, or split the application logic between them.

ii The interactions between these components (3 marks)

(1) The Controller receives the input (command) from the user and sends the message to the Model; (2) The Model performs the computation and sends the results to the View; (3) The View displays the results to the user.

In some version of MVC, the controller also handles events from the model and passes these to the view, so the model and view do not interact directly at all. This alternative was accepted providing it was clearly explained. As a number of people said, the controller may also update the view to show error messages etc.

iii The mapping from this architecture to the client-server computing architecture. (3 marks)

The View and Controller are mapped onto the client computers (Views = output logic; Controllers = input logic). The Model is mapped onto the server computers.

Two other options were also accepted. In some (fat client) cases the model may be split between the client and the server, for instance in ABC much of the business logic is on the client, while data storage in on the server. Secondly, in the case of a thin client (particular in a web application where the client is jut displaying HTML), the controller may be on the server.

A number of answers discussed tradeoffs, which was not the intent when the question was set but was ok provided the discussion was clear and concise. General discussions of client-server without relationship to MVC was not acceptable though - the word "mapping" in the question is important.

A number of answers said things like "the server is the model" rather than e.g. "the server holds the model". Clearly a physical component cannot be a software component.

Overall clearness and conciseness: **2 marks**

b). A doctor appointment system contains the following classes: Patient, Appointment, PatientTable, AppointmentTable, PatientUI, AppointmentUI, PatientDataAccess, and AppointmentDataAccess. Your task is to arrange these classes into a layered architecture.

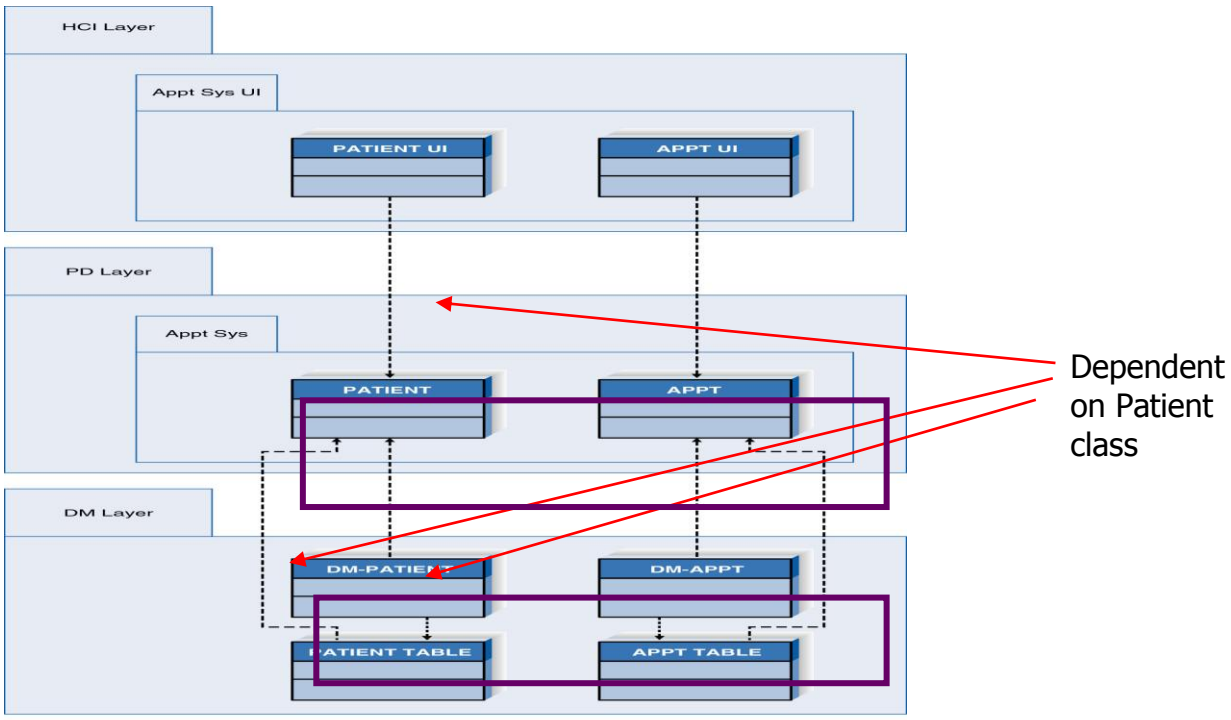
i State which class is placed in which layer (you do not need to explain why each class is placed where it is) (3 marks)

PatientUI and AppointmentUI classes are placed at the HCI layer; Patient and Appointment classes are on the problem domain layer; the rest are on the data management layer. (1 mark for each layer).

Since this very example was used in the lectures, it was remarkable how many people got it wrong, in particular many people split the data management layer into two (as shown in the lectures, it is normally considered two partitions, rather than layers.). Some people even got them in the wrong order, with the tables in the PD layer.

Draw a diagram which shows the dependency relationships between these classes and annotate it to briefly explain those relationships. (6 marks)

See diagram below. 3 marks for correct dependencies, 3 for correct, concise description.



This was a difficult question to mark, because the wording of the question allows a wide range of diagrams to be drawn. In many cases, the dependencies were drawn indiscriminately, with associations between layers in both directions (or with no direction shown). The key idea shown in the diagram above is that the pure fabrications in the first and third layers are dependent on the problem domain classes and not vice-versa. To see why, consider the following:

public class PatientUI {

private Patient _patient;

This makes sense because the PatientUI needs access to the Patient object it's displaying. Now let's try the opposite:

publc class Patient {

private PatientUI _userInterface;

Now this is reducing the cohesion and increasing the external coupling of the Patient class: it should only have those attributes which make it a Patient. In ABC for example, we have a package which contains the core problem domain classes such as Question and Answer. There are many dependencies on these classes but they depend on nothing outside the core package.

In other words, avoid two-way dependencies wherever possible. Another example of this is the relationship between the database tables and the wrapper classes. This is one-way because the wrapper classes need to "know" about the database tables but not vice-versa. Most solutions showed too many dependences, in a lot of cases every conceivable one.

SET BY: John Sargeant

QUESTION NUMBER: B2 (Problem-solving + Analysis).

LEARNING OUTCOMES ASSESSED: Applying GRASP design principles to a small design problem..

Question B2

a). Briefly explain the role of GRASP patterns 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..

Answers with examples are also good.

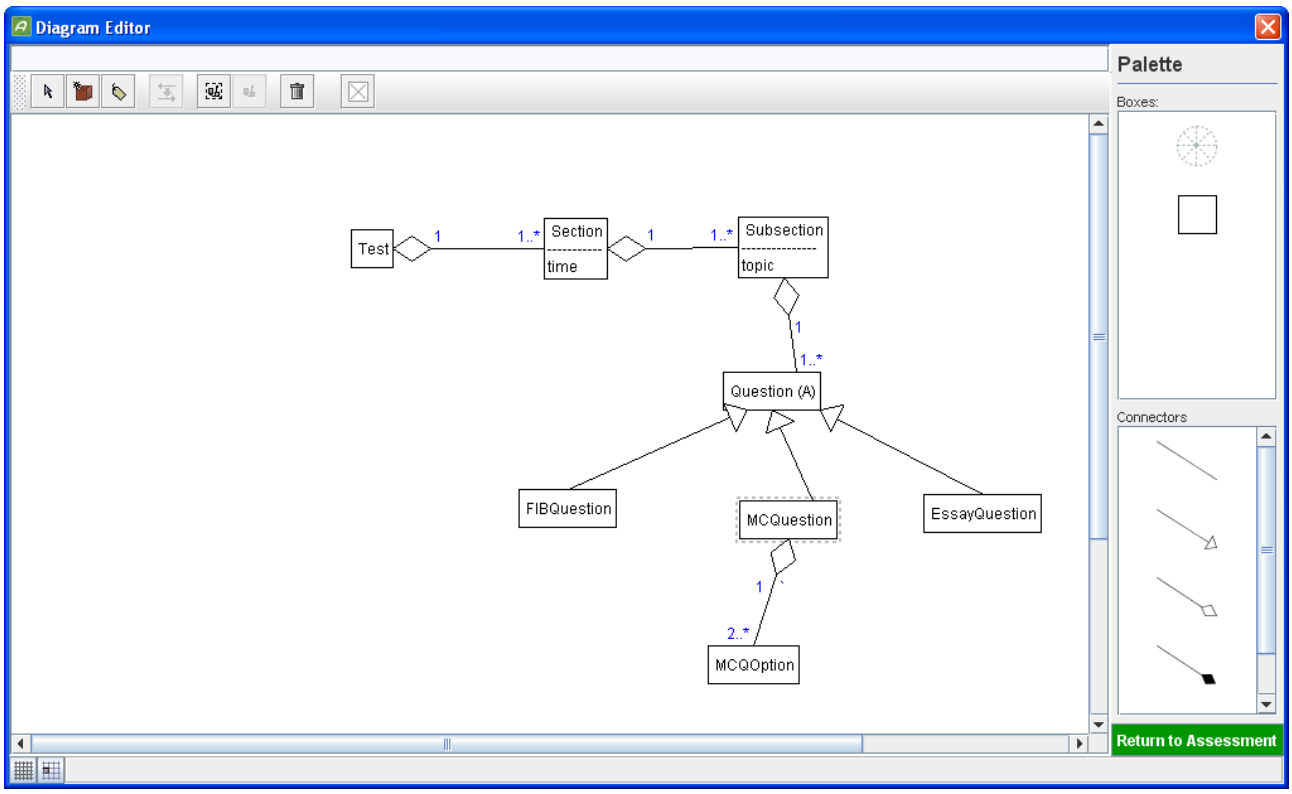
A number of people related GRASP to requirements change (following the principles makes it easy to deal with change) which was good.

Really everybody should have got full marks for this, but many answers failed to say two clear, distinct and relevant things. Very disappointing and probably reflects lecture attendance.

b). You are designing a system to test students in English language skills in order to find people who need to take remedial English classes. You have elicited the following requirements about the structure of the tests.

A test consists of one or more sections, each of which is timed separately. Each section contains one or more subsections, each of which is on a particular topic. Each subsection contains one or more questions. Questions are of three different types: Multiple choice questions, "fill in the blank" (FIB) questions and essay questions. Multiple choice questions have two or more options.

Draw a class diagram which BOTH captures all and only the information given in the above description AND leads directly to a design which adheres to GRASP principles. Hint: you should have exactly 8 classes, of which one is abstract. (6 marks)



Acceptable variations: different names for the types of question (provided they are appropriate singular nouns); using a plain line marks "contains" or similar instead of the diamond. For answers of the overall right shape deduct 1 mark for each of

- **Additional classes or attributes not mentioned in the description (or operations). Or any other extraneous clutter**
- **Missing classes or attributes.**
- **Abuse of UML notation**
- **Inappropriate names (e.g. "Sections" rather than "Section")**

Award max 2 for completely different solutions.

Most solutions had basically the right structure, but almost all committed at least one of the sins above, with unnecessary loss of about two marks on average. In particular there's not excuse for not using the tiny subset of UML class diagram notation presented in the lectures correctly.

c). For each of the following GRASP principles, explain how the design you showed is consistent with it. (2 marks each = 10 marks). Hint: if you cannot answer these questions you may wish to revise your design.

i High Cohesion

Each class represents a single well-defined entity, e.g. Question represents the notion of a question in general, which MCQ specifically represents MCQs.

Almost everybody understood high cohesion, but many people scored only one mark because they said only one thing (i.e. the first part of the sample answer above). Basic exam technique: for n marks you need to say n distinct things.

ii Low Coupling

External code is coupled only to the Test class so external coupling is low. The design has low coupling internally to, for example the Question subclasses are coupled only to Question (and not vice-versa).

Almost everybody understood low coupling. Most answers only mentioned internal coupling, which was ok provided they said two different things; again the problem was that many did not.

iii Information Expert

The design makes it easy to ensure that operations are put in the classes which have the information to carry them out. For instance the a getMarksAllocated() method would naturally goe in Question, and getCorrectOption() in MCQ.

Answers to this slightly harder question varied widely. Some were very good, others had no idea what Information Expert is, and several people did not attempt this part.

iv Polymorphism

Obviously polymorphism is used in the design in the form of Question and its subclasses. Question will contain the properties common to all question types while each subclass will contain properties specific to the relevant question type. This leads to the benefits of low coupling, high cohesion etc. described above and also…

There were a lot of good answers to this part. Worryingly, a number of people confused inheritance and contains relationships, stating that e.g. Section inherits from Test.

v Protected Variations

… protects against variation. If another Question subclass is added, or the functionality of one of them changes, the rest of the code is not affecting – in fact it doesn't even need to be recompiled.

Most answers were good, and only a few people didn't actually know what PV means.

d). Suggest two pure fabrications which could be added to the domain-inspired classes to help complete the design. (2 marks)

e.g. if the questions are stored in a database we will need a database connector to get them; we could generate questions from a factory (given a topic get a subset of the questions available on that topic). Marks for anything sensible.

Again, almost everybody knew what a PF is. As well as the two above UI classes were popular answers. Some answers were too vague or unrelated to the application to get marks, e.g. "A factory could be used to create objects."