THIS PAPER CONSISTS OF 2 SECTIONS, 14 QUESTIONS
Section A contains 10 short-answer questions, each is worth 2 marks (Total = 20 marks).
Section B contains 4 questions (Total = 40 marks).
Total: 60 marks.

CANDIDATES SHOULD ANSWER ALL QUESTIONS

Answers for Section A are to be written in the spaces provided on the exam paper.
Answers for Section B are to be written in the answer booklet provided.
Exam papers are to be collected with the examination answer booklets.

Sample answers are shown in blue. Put down any assumptions that you made if they help to explain your answers. Due to the time constraint in the exam, you don't need to write long assumptions as given in the sample answers. ***The text in red are additional explanation to the sample answers. They are not part of the sample answers. Du Huynh, April 2017

SECTION A

QUESTION 1 **[2 marks]**

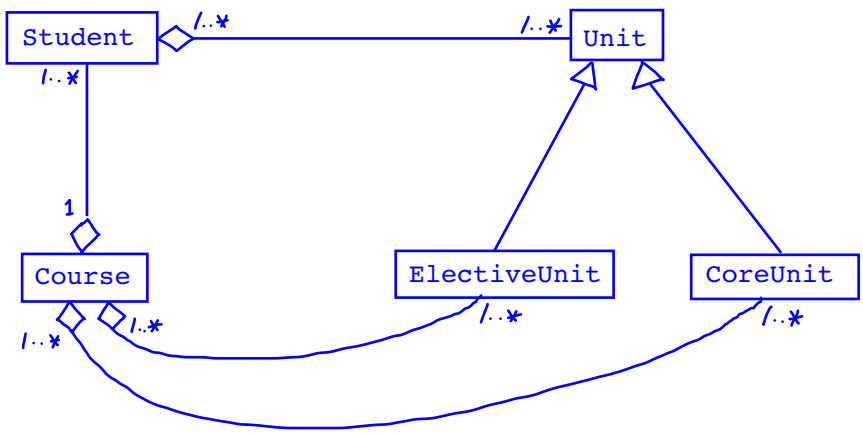
Explain how an agile software process differs from a waterfall software process and give an advantage and a disadvantage of each. (See lecture 1 and lecture 15 for this question)

A waterfall software process consists of a number of stages: project initiation, requirements analysis, design, implementation, testing, and installation. The process is linear, containing no cycle looping back to the previous stages.
An advantage: (Anyone of the following is sufficient)
* the process provides nice milestones (managers love the waterfall model)
* as the process is linear, there is no need to look back, i.e., one activity at a time
* it is easier to check progress (e.g., 90% coded, 20% tested)
A disadvantage: (Anyone of the following is sufficient. See lecture notes for more)
* software development is iterative - problems with requirements may be identified during design, problems with design may be identified during coding, etc.
* may end up delivering the wrong product as the client is involved only in the requirements elicitation stage.

An agile software process focuses on building software incrementally (or iteratively), with emphasis placed on the people rather than the process. It has the customer closely involved throughout the development process. It is therefore more suitable for business applications where the system requirements usually change rapidly.
An advantage: (See lecture 15 for more)
* the involvement of the customer helps to provide new system requirements, leading to more customer satisfaction to the final product.
A disadvantage: (See lecture 15 for more)
* for large projects, it is difficult to assess the effort required at the beginning of the software life cycle.

QUESTION 2 **[2 marks]**

Draw a UML class diagram to capture the following situation: "Every student is enrolled in a course. Each student may be enrolled in a set of units. Some units are core units for one or more courses and some units are elective units for one or more courses."



Assumptions: a Course must have at least one Student enrolled in it and, as every student is enrolled in a Course, the relationship between Course and Student is 1-to-many. Assuming that elective units can be shared by different courses, between Course and ElectiveUnit is a many-to-many relationship (Similarly between Course and CoreUnit).

QUESTION 3 **[2 marks]**

Identify the **actors** and the objects in the following scenario to register a patient in a hospital management system:

The administrator enters the patient's name, address, date of birth and emergency contact details into the system. If the patient has only public health insurance, the administrator enters the patient's medicare number, and the system verifies this with government health database. If the patient also has private health insurance, then the administrator enters also the patient's private health insurance details, and the system verifies these details with the private health insurance system. When these details are verified as correct, the system saves the patient's details and confirms the registration.

Actors:
Administrator, Government Health Database, Private Health Insurance System
(The last two are external systems)

Objects:
Patient
Administrator
Address
EmergencyContact
PublicHealthInsurance
PrivateHealthInsurance
Registration

Note: we focus a lot on OOP in the Design part of CITS4401. In the Object Modelling part of the unit, we try to identify objects (and thus classes). So if the data needs to be stored away, then OODB would be more suitable. Please do not mix up with relational DB, spreadsheet, and SQL that you learn from other units. While OOP might not be your favorite subject, it is important to know how to identify entity objects for a given problem.

Assumptions:
* Address contains street number, street name, suburb, and postcode and so is large enough to be an object.
* Other nouns identified that are not considered to be objects include: patient's name and date of birth. They are considered to be stored as primitive types or strings and are attributes of other objects.
* EmergencyContact is assumed to contain a contact person's name and phone number (and most likely the address and relationship to the patient) and so should be an object rather than an attribute.
* the noun "details" appears a few times. It is not considered to be an object as the word refers to the attributes of some objects.

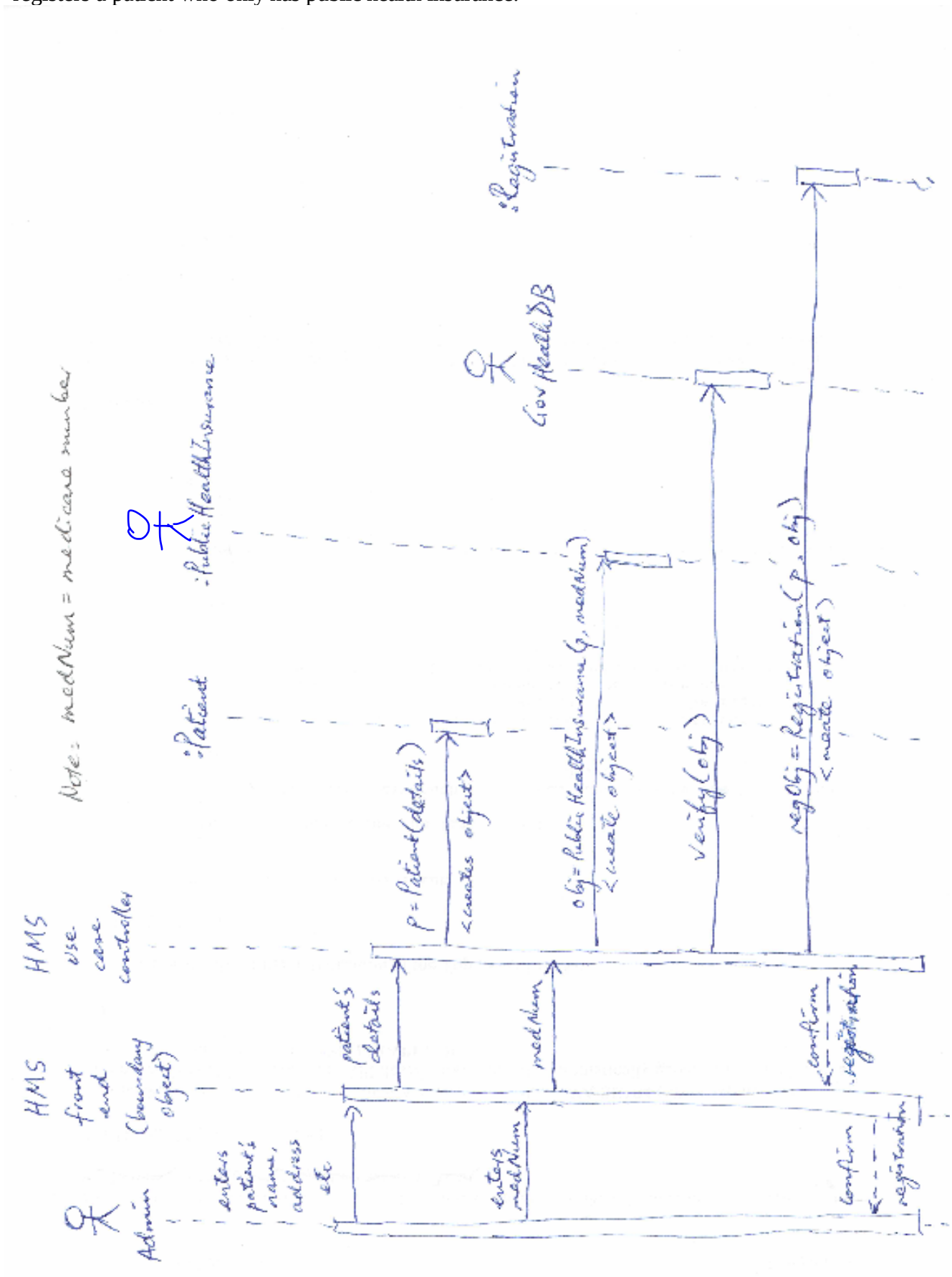
Note: you may have different assumptions from the above.

Comments:
* Highlighted in yellow are actors.
* Other nouns are highlighted in pink. Some of them (e.g., name, date of birth) are obvious attributes of other entity objects. Address is large enough to be an entity object as it can include street number, street name, suburb, and postcode. You should provide a brief statement about your assumption to some of the nouns if it is not clear or obvious to you whether they should be attributes or objects. A rule of thumb is: if the noun is small enough to be represented by a primitive type (int, float, char, etc) or a string (char array) then it should definitely be an attribute; otherwise it should be an object.
* The word "system" is a noun but should not be an object as it refers to the whole hospital management system.
* Don't forget to include Registration as an object in your answer. This is the most important entity object. At the end of the registration process, the system should generate a Registration object that records all the details such as a registration number, the date the registration takes place, a reference to the Patient object, a reference to the patient's public health insurance etc. Although these details are not given in the description, it should be clear that Registration is an entity object.

See page 5 for comments for Q4.

QUESTION 4 **[2 marks]**

Refer to Question 3 and sketch a UML sequence diagram for the scenario where the administrator registers a patient who only has public health insurance.



QUESTION 5 **[2 marks]**

Describe the **layered** software architecture, and give one advantage and one disadvantage of it.

The layered software architecture has the layers organized in a hierarchical fashion. Each layer provies a service to the layer above and acts as a client to the client below. The layers may either be partially opaque (i.e., some interaction between non-adjacent layers are allowed) or completely opaque (i.e., only adjacent layers can interact)

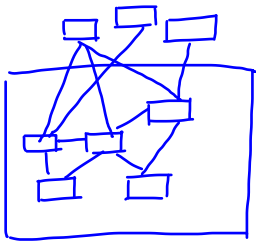
Advantage: (only one advantage is needed. Any one of the following would suffice)
* the architecture allows one to construct the design of a system based on increasing level of abtraction. So complex problems can thus be solved with a sequence of incremental steps.
* the architecture supports enhancement, as changes to one layer affect only the two adjacent layers.
* the architecture supports reuse. Different implementations of the same layer can be used interchaneably if they support the same interface.

Disadvantage: (only one advantage is needed. Any one of the following would suffice)
* not all systems can be easily structured in layers. The architecture is therefore only applicable to a small number of problems.
* the architecture may need close coupling between the logically high-level functions and their low-level implementations.
* it can be difficult to find the right level of abstraction, e.g., finding the protocols which bridge several layers.
* there might be a negative impact on the performance of the system as we have extra overhead of passing through layers instead of calling a component directly.

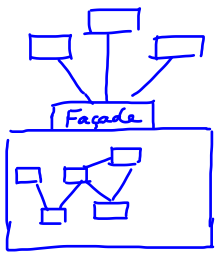
QUESTION 6 **[2 marks]**

Describe the Façade design pattern. Give an example to show when it would be suitable to use this pattern.

The Façade design pattern provides a unified interface to a set of objects, thus allowing the detailed implementation of these objects to be encapulated in a closed architecture. With a closed architecture, the subsystem can decide exactly how it is accessed.



Open architecture (without Façade)



Closed architecture (with Façade)

An example: it would be suitable to use the Façade design pattern for each subsystem in a large system. Having the components of the subsystems hidden behind the unified interface helps to reduce the coupling between subsystems.

QUESTION 7 **[2 marks]**

Rewrite the following as a structured rationale argument: "The main issue is what mathematics library to use for the *Warehouse-Stock* subsystem. We could use the NAG library which supports multiple programming languages including C, C++, Java, and Fortran. However, no one in the project team is familiar with this library. So a 2 weeks' learning period will be necessary to learn the library. From a previous project, some team members have used ALGLIB, which is an open source numerical analysis library that may be used with C++, C#, and VBA. From past experience, ALGLIB is known to be not reliable at times. As *Warehouse-Stock* needs to interface with other subsystems already implemented in Java and to avoid the reliability problem with the ALGLIB, the project team decided to allocate learning time and use the NAG library."

```
Issue: What mathematics library to use for the Warehouse-Stock subsystem?

Proposals:
P1: use the NAG library
P2: use the ALGLib

Arguments:
P1:
  A+ supports multiple programming languages including C, C++, Java, and Fortran
  A- no one in the team is familiar with this library so a 2 weeks' learning period
     is required
P2:
  A+ is an open-source numerical analysis library
  A+ may be used with C++, C#, and VBA
  A- is known to be not reliable at times

Criteria: the Warehouse-Stock subsystem needs to interface with other systems
already implemented in Java.

Resolution: proposal P1 is chosen given the criteria and to avoid the reliability
problem with the ALGLIB, the project team will allocate learning time and use the
NAG library.

Comments:
* It is fine if in your answer you combine the two A+'s under proposal P2 together to
form one A+.
* Make sure that you structure your answer under the 5 headings: Issue, Proposals,
Arguments, Criteria, and Resolution.
```

```
Comments for Q4: you can use shorthands where appropriate, e.g. I use medNum to denote
'Medicare number' and HMS to denote 'Hospital Management System' in the sequence
diagram. I hope that it is obvious that various objects would be created in the
registration process, e.g., a Patient object is created (in Java, the constructor that
creates objects has the same name as the class name), a PublicHealthInsurance object is
created using the Patient object and the Medicare number as parameters. I put down Java
constructor calls as events in the diagram, but if you just put down 'creates Patient
object', that would be fine in the exam. In the registration process, the controller
needs to verify the  PublicHealthInsurance object with the Government Health DB (an
external system), we can simplify the interaction by omitting the boundary object at
that end (as it is outside the scope of HMS). You may also insert dashed arrows back to
the controller to indicate that verification was successful, registration object was
created successfully, etc., but you probably won't have the time to do so in the exam.
Make sure that (1) you have a controller controlling the use case (2) access to any
entity objects is always via the control object.
```

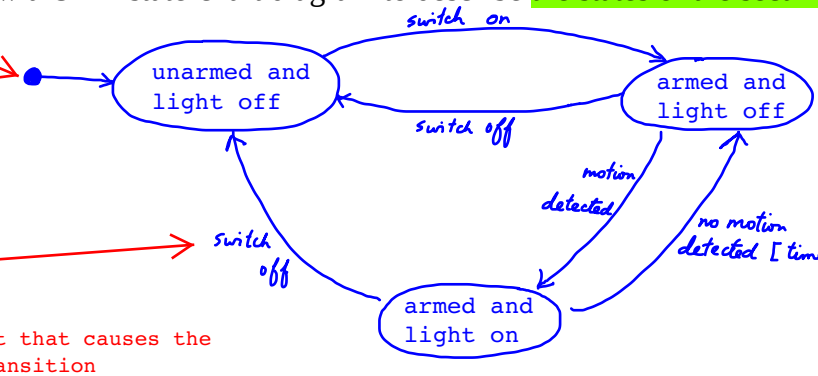
Make sure that you show which state is the starting state.

You are asked to draw a state-chart diagram to describe the states of the security light system, *not* the states of the switch. So your diagram should not have states such as "the switch is on" and "the switch is off".

QUESTION 8 **[2 marks]**

A security light system has a switch and a motion sensor attached. It can be either *armed* or *unarmed*. If the switch is in the *off* position the light is off and the system is unarmed. When the switch is turned on, the light stays off but the system is armed. If the system is armed and the motion sensor detects movement, the light comes on. If no movement is detected for 5 seconds, the light goes off.

Draw a UML state-chart diagram to describe the states of the security light system.



a guard that is associated with an event must be a Boolean expression. State transition takes place only if the expression evaluates to True.
Make sure that you put the expression inside square brackets.

The event that causes the state transition

Comments: you might have drawn two state-chart diagrams - one diagram for the "light on" and "light off" states, and another diagram for "unarmed" and "armed". However, with two separate state-chart diagrams, it is not clear how they relate to each other. For instance, what event would lead to the transition from "light off" to "light on"? - turning the switch on does not turn on the light; the light is turned on only if motion is detected by the sensor; however, at the "light off" state, we don't know whether the system is armed or unarmed. Whenever you have ambiguity like this, it is better to put the two diagrams together and work out meaningful names for the states so that there is no ambiguity. Also, in the question, you are asked to draw 1 state-chart diagram, not 2 state-chart diagrams.

QUESTION 9 **[2 marks]**

Briefly describe what should be stored in a traceability table.

See lecture 20.

Traceability tables are used to store requirements and all their related information. In a traceability table, each requirement should have
* a unique requirement identifier
* important customer observable features
* source of the requirement
* dependency of the requirement
* subsystems governed by the requirement
* interface of the requirement

QUESTION 10 **[2 marks]**

Give three reasons why requirements negotiation is needed in software engineering.

Reasons: (any three of the following)

* Negotiation helps to encourage communication and therefore improve understanding of the requirements.
* Negotiation helps to reveal conflict, solution exploration, and collaborative resolution.
* Negotiation helps to improve the agreement level among the stakeholders.
* Negotiation helps to develop stakeholders' satisfaction.
* Negotiation helps to improve requirements quality.

SECTION B
Answers for Section B are to be written in the examination answer booklets.

All Questions in Section B refer to the *Remote Pair Programming System:*
Pair programming is an agile software development technique in which two programmers work together at one work station. One types in code while the other reviews each line of code as it is typed in. The person typing is called the driver. The person reviewing the code is called the observer. The two programmers switch roles frequently (possibly every 30 minutes or less).

Suppose that you are asked to build a system that allows *Remote Pair Programming*. That is, the system should allow the driver and the observer to be in remote locations, but both can view a single desktop in real-time. The driver should be able to edit code and the observer should be able to "point" to objects on the driver's desktop. In addition, there should be a video chat facility to allow the programmers to communicate. The system should allow the programmers to easily swap roles and record rationale in the form of video chats. In addition, the driver should be able to issue the system to backup old work.

QUESTION 11 **[12 marks]**

a) (4 marks) Draw a use case diagram to show all the functionality of the system.

b) (4 marks) Describe in detail four non-functional requirements for the system.

c) (4 marks) Give a prioritized list of design constraints for the system and justify your list and the ordering.

QUESTION 12 **[12 marks]**

a) (6 marks) Describe a software architecture that would be suitable for the system.

b) (6 marks) Present a structured rationale argument for your software architecture using the design constraints that you identify in Question 11 c) above.

QUESTION 13 **[12 marks]**

a) (7 marks) Propose a set of classes that could be used in your system and present them in a class diagram. **Note:** You only need to include entity objects in your class diagram.

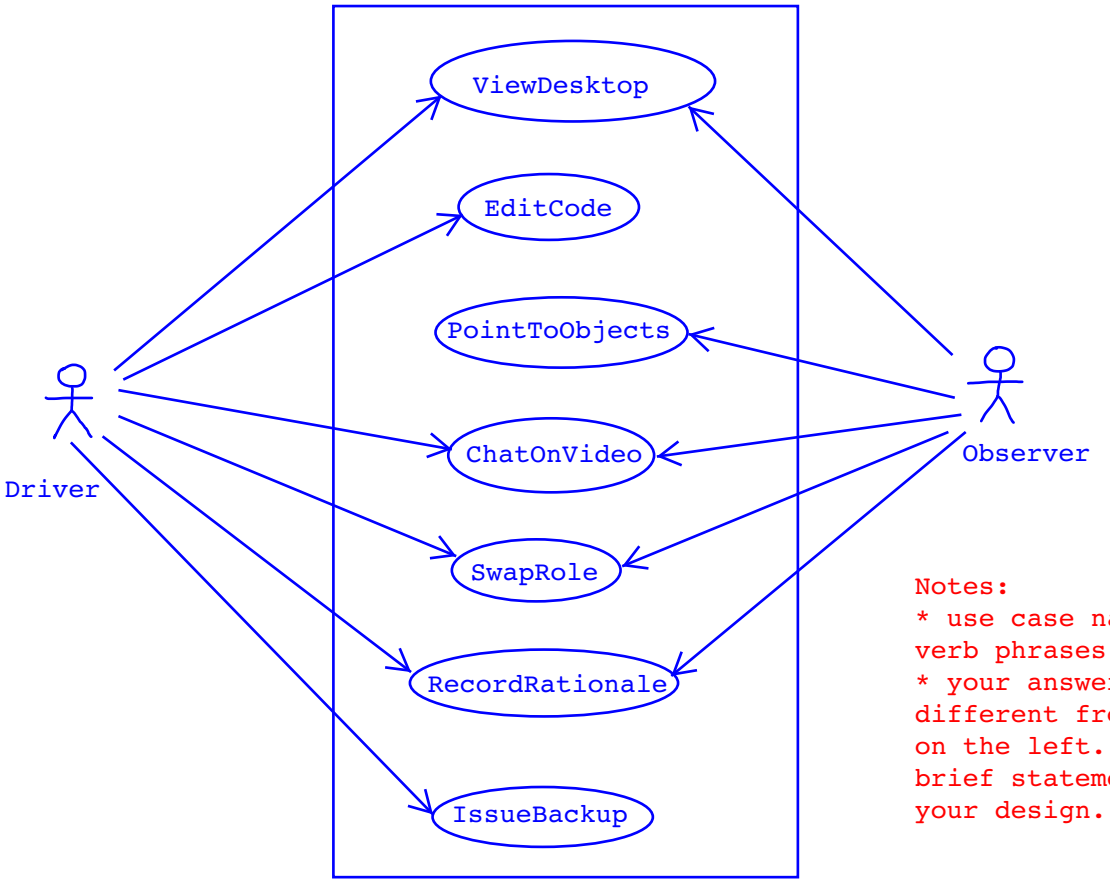
b) (5 marks) Propose a subsystem decomposition for these classes and comment on the coupling and cohesion within this decomposition.

QUESTION 14 **[4 marks]**

Identify two design patterns that would be suitable for the system. Briefly explain your answer.

---------- **END OF PAPER** ----------

Question 11 a)



Notes:
* use case names should be verb phrases.
* your answer may be slightly different from what is shown on the left. Please put in brief statements to explain your design.

Assumptions:
* when the Driver edits code, we assume that the Observer can see the changes in real-time through the ViewDesktop use case, thus there is no arrow pointing back to the Observer for the EditCode use case. A similar assumption is made for the PointToObjects use case, so no arrow points back to the Driver.
* we assume that both the Driver and Observer can initiate the ViewDesktop, ChatVideo, SwapRole, and RecordRationale use cases.

Question 11 b)

Any four of the following would be sufficient as an answer. You may have additional NFRs that are not on my list below. Make sure that you supply explanation to each NFR that is related to the system (rather than just "Ease of use", "Reliability", "Security", etc, which are generic NFRs that are desirable to most systems).

* Ease of use - the front-end interface must be simple and easy to use.
* Real-time performance - the Observer should be able to see the changes made by the Driver immediately without delay; the video chat should be smooth without delay also.
* Availability - the system should be available to both programmers all the time.
* Portability - the programmers should be able to use the system regardless of what computer and operating system used by the programmers.
* Security - the backup code should be kept securely and be protected from unauthorized access.
* Cost - users should pay no more than $5 per month to use RPP.
* Reliability - the system should be reliable, i.e., it should not crash when the internet speed is slow and when the internet connection is suddenly down the user should be able to resume the session at a later time.

Question 11 c)

Note that since you need to supply only 4 NFRs in part b), you therefore only need to specify your ordering of the corresponding design constraints.

In your answer, you should explain your ordering, e.g., if you put "ease of use" in position 1 and "real-time performance" in position 2, then justify why you think "ease of use" is more important than "real-time performance". You should also discuss briefly whether there are conflicting design constraints as well and, based on your ordering, which design constraint should be emphasized on.

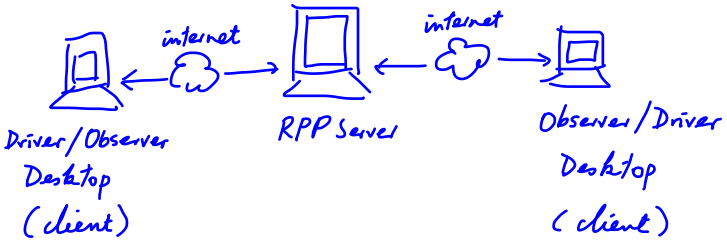
Note: NFRs become constraints on the design and, as a result, "design constraints" are often treated the same as "NFRs". However, there is a difference between them.

Example 1: "the system should be portable" is a NFR. This NFR may lead to a constraint on the programming language used for the implementation of the system (e.g., the programming language Java (rather than C and C++) might be preferred in order to meet this NFR).

Example 2: "security - the system must be secured" is a NFR. The design constraints could be a user authentication must be in place, the communication protocol must be encrypted, and/or the data must be stored on a server behind firewall.

Question 12 a)

A client-server architecture would be suitable for RPP. The server should hold the data, including different versions of the programming code, rationale documents, video clips, and session information. Both programmers are clients of the system. The components of this architecture are the server and the desktops of the two programmers. These components interact through the Internet. The communication between the server and each desktop is two-way. The two clients do not directly communicate with each other.



You can expand the paragraph above, taking into account the NFRs that you have identified in Q11.

At ether other software architecture more suitable?

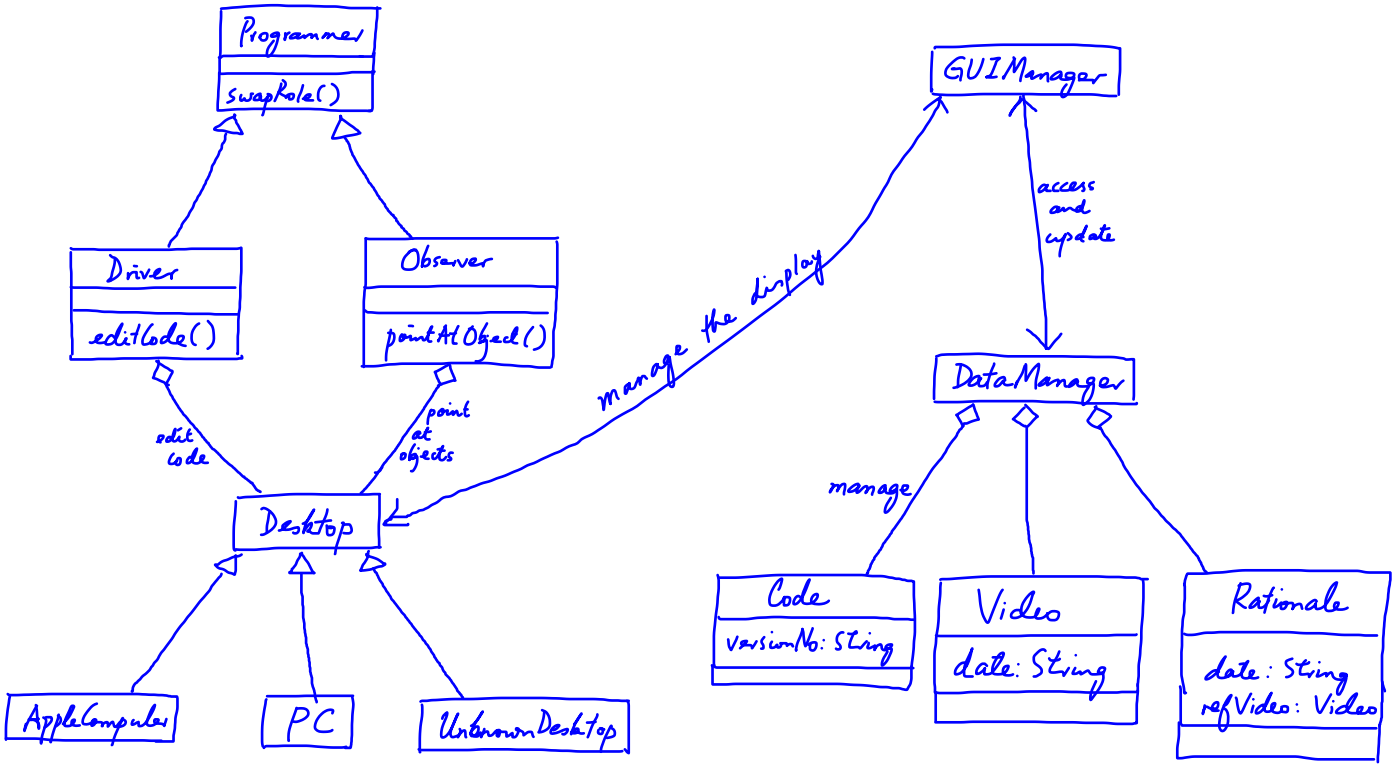
Question 12 b)

This is a very open question. You should write a structured rationale argument as in Q7 for your architecture (your answer in Q12a).

Issue: Which architecture should we use for RPP?
Proposals:
P1: ...
P2: ...
...
For the criteria, you should refer to the design constraints from Q11c).

When answering Q12a), you must have already considered all the software architectures that you know. How did you come to the resolution?

Question 13 a)



*** Give a brief explanation about the diagram.

The sample class diagram above captures most of the classes and relationships. I have inserted an UnknownDesktop class for future extension of RPP. Obviously when I designed this class diagram I already have the AbstractaFactory design pattern in my mind.

In your class diagram you don't need to mention attributes and operations except for those really important ones. Some of the operations can be extracted from the use case diagram (Q11a)). For instance, you can probably spot very quickly that, under the Driver class, I should have the issueBackup() operation. Also, under the Programmer class, I should have the viewDesktop(), chatOnVideo(), and recordRationale() operations. Maybe we should have Session as a class as well and associated with DataManager? I left it out because it is not explicitly mentioned in the description. Note that since AppleComputer and PC are not explicitly mentioned in the description, if you don't have any subclasses under Desktop it is fine as well.

A second version is to have the Programmer class associated with the Desktop class instead. However, in this second version we cannot label the association whether it is 'edit' or 'point at objects'.

The GUIManager class and the DataManager class (both can be interfaces) are inserted to help manage the display and the access of data. The model-view-controller (MVC) software architecture pattern is adopted here: DataManager takes care of the 'model', the GUIManager, which interacts with DataManager and Desktop, takes care of the 'view', and the operations carried out by the two programmer compose the 'controller' part.

Question 13 b)

One obvious subsystem decomposition is to have 3 subsystems separating 'model', 'view', and 'controller'. So,
* Programmer and its two subclasses in subsystem 1.
* Desktop and all its subclasses together with GUIManager in subsystem 2.
* DataManager and its subclasses in subsystem 3.

An alternative is to have 4 subsystems with GUIManager in one subsystem by itself.

Exercise: comment on the coupling and cohesion.

Question 14

As described earlier, one design pattern is the AbstractFactory pattern. We can use this pattern to create all the different desktop objects. If new desktops appear in the market and need to be supported by RPP, the AbstractFactory pattern will take care of that seamlessly.

Another obvious design pattern is Facade. This should be obvious in the class diagram, e.g., the GUIManager does not directly access Video, Code, etc; instead, these specific objects are hidden under the DataManager class, which has the role of a Facade.

Note that we use the MVC software *architecture* pattern in Q13 but it is not a design pattern (which is smaller).

Can you think of other design patterns? Perhaps the Bridge pattern? Maybe in the future we want to use the RPP system as a tool to teach students about remote pair programming and we want the students to be represented as a PassiveObserver class who can view the desktops of both Driver and Observer? The Bridge pattern would help the expansion of class of users.

How about the Proxy design pattern? Any other patterns might be suitable?