2016

Q1 Software Engineering

1.Not every code development counts as "software engineering". List three aspects that are characteristic of true "software engineering" projects.

Software Engineering is programming under at least one of these two conditions:

more than one person writes and uses the program

more than one version of the program is created

Software Engineering is the application of principles, abilities and craftsmanship on the design and the construction of program systems

Software Engineering is the practical application of scientific rationale on the design and the construction of program systems

2.There are four factors of software development of which the client may prioritize three, By improving what property of the software development process can one gain an advantage regarding all four factors? Briefly point out why this is the case for two of these factors.[6 marks]

Reliability, maintainability, reusability, automation

Productivity Enhancers

•High Reuse » using parts multiple times

•Good Maintainability » fix shortcomings » extend functionality » address changing

3.Is there a difference between "building the right (desired) system" and "building the system right (properly)"? Briefly explain your answer using corresponding software engineering terminology.[6 marks]

Aspects of Reliability

**Correctness** is defined as the conformance of the system to its specification

“Are we building the right system?”

**Robustness** is defined as the ability of a system to (continue to) perform despite being forced to operate outside specified parameters

“Are we building the system right?”

4.Briefly explain what the term "maintenance" means in software engineering and what typical activities are performed in the maintenance phase.[6 marks]

Maintainability

How easy or hard is it to detect and correct errors in a system? How easy or hard is it to change the system?

Issues back then programming in the large, system structure, error propagation and change avalanches

» fix shortcomings » extend functionality » address changing

» error correction („right“, ca. 20%)

» change of construction („better“, ca. 20%)

» change of specification („different“, ca. 60%)

1. 3Briefly discuss the meaning and significance of "maintenance" in software engineering.

[4 marks]

How easy or hard is it to detect and correct errors in a system? How easy or hard is it to change the system?

Issues back then programming in the large, system structure, error propagation and change avalanches

» error correction („right“, ca. 20%)

» change of construction („better“, ca. 20%)

» chIn computing, an interface is a shared boundary across which two separate components of a computer system exchange information.ange of specification („different“, ca. 60%) 36-64

Maintenance is the continuous updating of a system to fix shortcomings,

extend functionality and address ongoing changing requirements of the client.

Maintenance includes any change to the software after the product has been released and

it is estimated 60%-80% of a software's lifetime is spent in maintenance.

Maintenance is important in software engineering because software projects are rarely released without bugs which need to be fixed later.



1. What are symptoms of a software system that is hard to maintain? [5 marks

It's hard to maintain quality because software is often more complex than we realize. It's complex because 1) the problems it tries to solve are complex and 2) the methods involved in solving the problems are complex. That complexity is hard to get right, especially when technologies, requirements, staff turnover, etc. are unstable.

1. Fewer interfaces are considered better than many interfaces in software engineering. Briefly explain why this does not imply that zero interfaces are optimal. [5 marks]

In computing, an interface is a shared boundary across which two separate components of a computer system exchange information. Software interfaces provide access to computer resources (such as memory, CPU, storage, etc.) of the underlying computer system. So, interface is necessary in software engineering, i.e. interface cannot be zero. If it is, we do not have access to computer resources.

Briefly discuss the potential benefits and dangers involved in reusing software compo­nents. [6 marks

1. [4 marks] What are the four factors in software development of which the client may prioritize a maximum of three?

Reliability, maintainability, reusability, automation

[2 marks] Why do you think 40 years of software engineering practice and research have not been able to lead software development out of the above mentioned crisis?

Software systems are always more costly and require more time to build than planned. Moreover, reliability and correctness are rarely impeccable. » finish late (up to a factor of 2)» become too expensive (up to a factor of 10)» are cancelled because of the above

(c) [4 marks] Briefly explain the terms “Correctness” and “Robustness” making sure the difference between them is clear.

Aspects of Reliability

Correctness is defined as the conformance of the system to its specification

Robustness is defined as the ability of a system to (continue to) perform despite being forced to operate outside specified parameters

(d) [10 marks] Describe the steps a software engineer can take to minimise the possibility of software project failing using at least two of the examples discussed by at least two different group project presentations in the class.

Review and testing are an essential practice when writing code; especially high-risk code.

Version control is a software practice used by teams of software developers to ensure working pieces of code aren't lost or overwritten while someone else is editing the code

Automatic deployment is an effective method when updating elaborate systems with new functions.

Error prevention, traceability and alerts are useful tools to mitigate the damage of a fault as quickly as possible. Knights Trading Loss #Toyota ETCS

(b)The maintainability of a component correlates with the size of its interface. Briefly describe this correlation and mention two technical properties that components with the desirable interface size will typically exhibit.（2）

Interfaces » few, small, and explicit

Every module should communicate with as few others as possible.

(c) If a software system is hard to change because any change may break the system in some way, what is the system suffering from and what system property could address the problem? [4 marks]（2）

(d) Briefly discuss the potential benefits and dangers involved in reusing software components.（2）

Advantages:-

1.    It can reduce the overall cost of software development as compared to other model.

2.    It can reduce the risk. 3.    It can save the time of software development. b/c testing of component is minimize. 4.    Faster delivery of software.

Disadvantages:-

1.    Reuse-oriented model is not always practical in its pure form.

2.    Compromises in Requirement may lead to a system that does not meet the real requirement of the user.

3.    Organization using the reusable component, are not able to control the new version of component, this may lead to lost control over the system evolution.

Q2 Design Principles

1.Why are classes with low coupling desirable? Tick only one box.[2 marks]

□They increase cohesion.□They decrease cohesion. □ They increase rigidity. □ They decrease rigidity.

Rigidity It is hard to change because every change affects too many other parts of the system

2.Assume module A has five components with six connections and module **B** has five components with 8 connections. Which module is more likely to contain components better suited for reuse? Briefly explain your answer?[4 marks]

module A is more likely to contain components better suited for reuse. A has less connections than B, and it has fewer the interface count.

if a system is composed of n modules, the number of connections should remain much closer to the minimum, n–1 (C), than to the maximum, n (n – 1) /2 (B).

Highly coupled classes are difficult to reuse as they dependent upon many other classes

3.Which - "layers" or "partitions" - are useful to address modular continuity? Explain your answer.[6 marks] .(2)

Continuity» avoiding the "change avalanche"

How to reduce the interface count?

Layering » layers only interact with adjacent layers, stops change avalanches, separates concerns

4.Assume an implementation has been verified to be correct with respect to the specification. What other properties of the implementation may the customer be interested in?

5.Briefly list the pros and cons of pre-conditions and post-conditions for the purpose of achieving modular protection. [6 marks]

Protection » avoiding "error creep"

Pros: : A method satisfies Modular Protection if it yields architectures in which the effect of an abnormal condition occurring at run time in a module will remain confined to that module, or

cons : at worst will only propagate to a few neighboring modules.

preconditions: checking the validity of input data before it is used

6. Briefly explain the relationship between coupling and information hiding.(2)

Information hiding refers to the idea of selecting a subset of information within a module and classifying it as the official public information made available to client modules. Reducing the amount of information made public means that the module is likely to be depended on less by other modules and so high reliance on the module and its functions (coupling) is avoided.

(c)Why do internal criteria for a software system matter, if the client is only concerned with external criteria?

External Criteria (what clients expect) » correctness, robustness, efficiency

Internal Criteria (regarding solution) » extensibility, modularity ...

Only external factors really matter. But the key to achieving them are the internal ones

The internal criteria of a software system determines how easy the software is to understand and maintain. Internal criteria includes Rigidity, Fragility and Immobility. For example, if the client would like to change or add a new part to the external criteria, the developers must be able to go back and modify the internal criteria without breaking the rest of the software. It is very hard for the developers to do this if the internals are fragile, meaning if you make a change, unexpected parts will break.

(d) What causes a software system to exhibit "rigidity"?

Rigidity It is hard to change because every change affects too many other parts of the system

1. What causes a software system to exhibit "fragility"?

Fragility When you make a change, unexpected parts of the system break

1. [5 marks] What is meant by a “System” in Software Engineering? Please describe what it is and give some examples.
2. [5 marks] What is a system boundary? Please describe what it is and give some examples.
3. [5 marks] So-called control-objects manage other, comparatively passive, objects. Discuss the pros and cons of such “manager” objects.
4. Why are classes with high cohesion desirable?

High cohesion: a class has lightweight responsibilities in one area and collaborates with other classes to fulfil tasks

1. Which of the five modularity requirements that were discussed in lectures can help to improve continuity? Briefly explain your answer.

Decomposability » how to separate

Composability > how to combine

Understandability » if you want to change, you need to know what

Continuity » avoiding the "change avalanche"

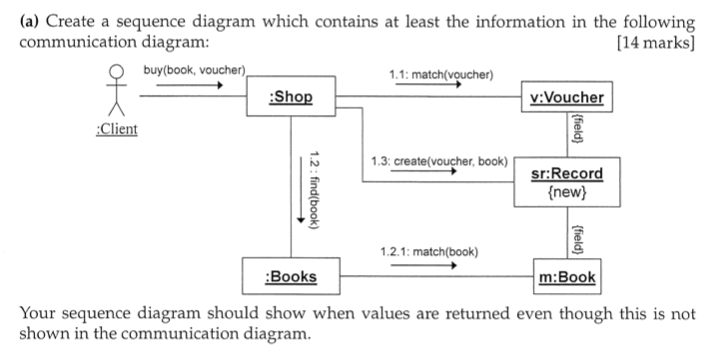
Protection » avoiding "error creep"

(d) Briefly explain why even a correct implementation does not guarantee full customer satisfaction and why this circumstance is not used to change the traditional development process.

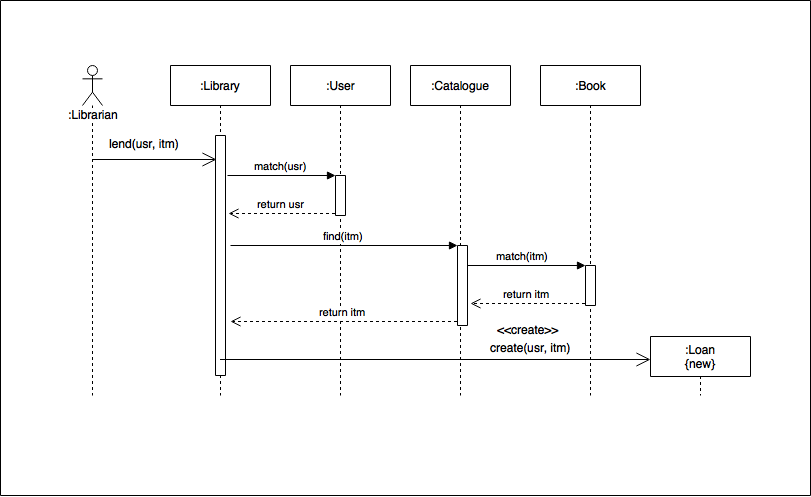
(e) [4 marks] Briefly discuss which of ”using pre-conditions” or “using post-conditions” are a better means to achieve “modular protection”.

pre-conditions is a better approach to achieving modular protection because it checking the validity of the input data before it use.

Q3 Interaction Diagrams

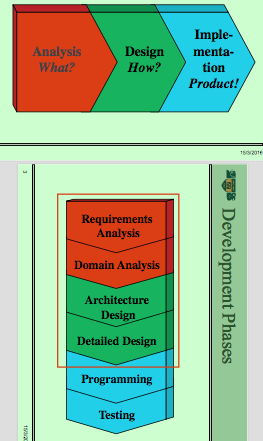
（2）

------------------------------------------------------------------------------------------（2）



---------------------------------------------------------------------------------------------------

1. In which development phases can one use interaction diagrams? Briefly describe the purpose of interaction diagrams for each phase you name. [6 marks]

Detailed Design-> interaction diagram 

during analysis, to improve individual or group understanding of inter-object behaviour

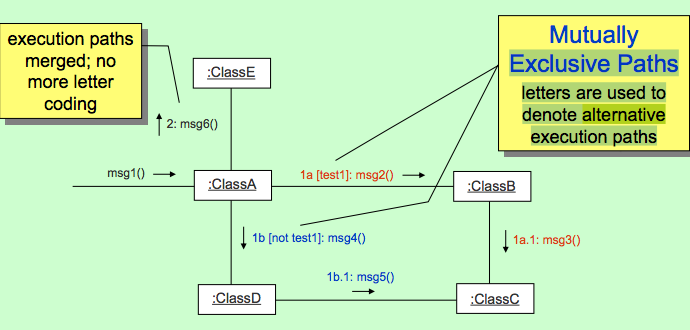
during design, to precisely (but typically partially) describe inter-object/process communication

during testing, the traces can be compared with those described in the earlier phases

1. In what way can you capture alternative execution paths in a communication diagram?

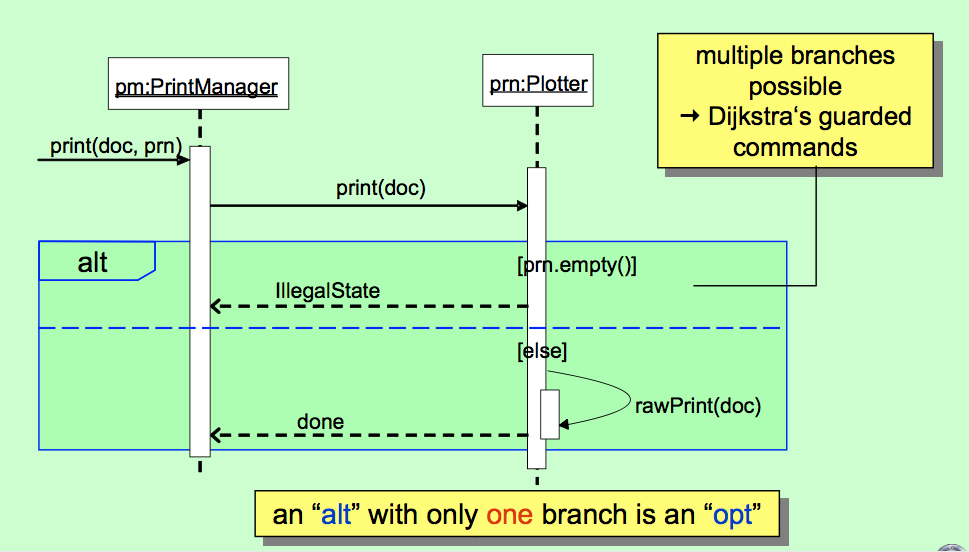
[2 marks]（3）Conditional Paths -> Mutually Exclusive Paths letters are used to denote alternative execution paths

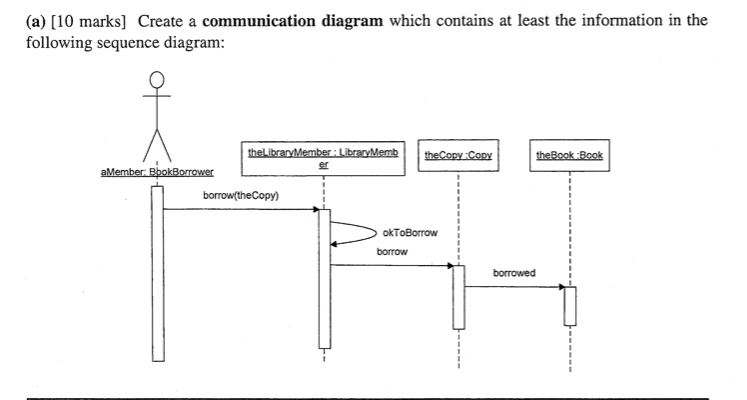
By using letters in the labelling of actions e.g. 1.1, 1.1.1, and include message guards e.g. [valid], [invalid] to specify which branch should be followed and when.



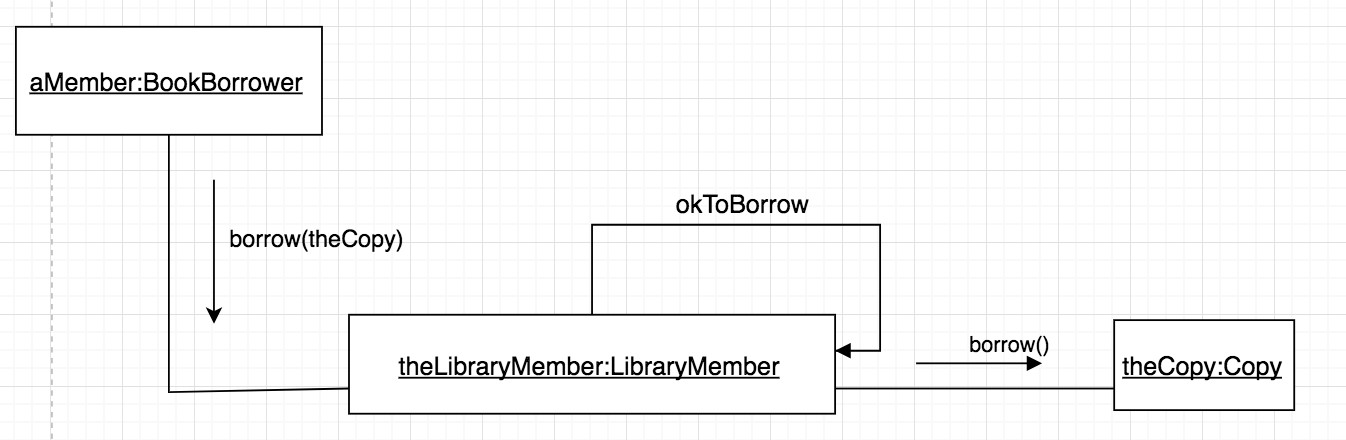
1. In what way can you capture alternative execution paths in a sequence diagram?（2）

an “alt” with only one branch is an “opt”





------------------------------------------------------------------------------------------



------------------------------------------------------------------------------------------

b) [5 marks] Compare and contrast sequence diagrams and communication diagrams. Discuss the advantages and disadvantages of both kinds of diagrams.

Sequence diagrams -> Strength: clearly show ordering of messages

Weakness: don’t show links become very wide

Communication diagrams: -> Strength: show links & use space economically

Weakness: difficult to see message sequence

1. Briefly explain how you could use interaction diagrams in both implementation and testing phases respectively（2）

during analysis, to improve individual or group understanding of inter-object behaviour

during design, to precisely (but typically partially) describe inter-object/process communication

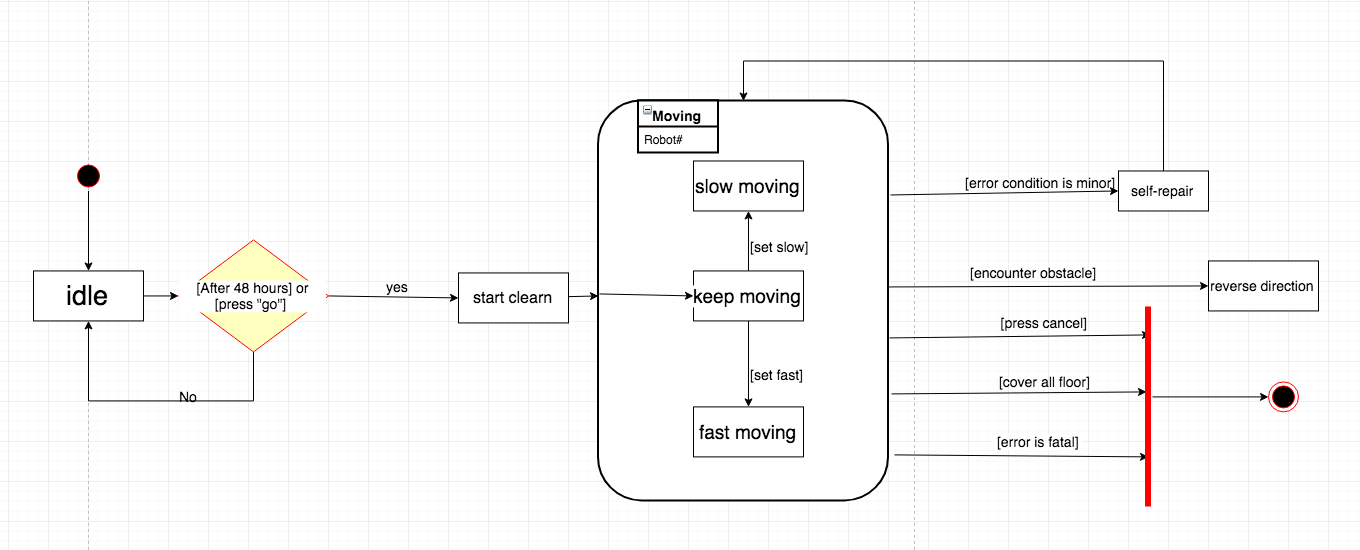
during testing, the traces can be compared with those described in the earlier phases

Q4 State Diagrams

1. Create a UML state diagram that describes the behaviour of a cleaning robot. Initially, the robot is idle. After 48 hours of inactivity or when the user presses the "go" button on a remote control, the robot starts cleaning the floor and normally keeps moving forward until it has covered all of the floor or the user presses the "cancel" button. The user can also set the speed to "fast" (noisier operation) or "slow" (quieter operation) at any time. If the robot encounters an obstacle, it turns by 15 degrees and then reverses direction - i.e., moves backwards if it has been moving forwards and moves forwards if it has been moving backwards. At any point in time an "error" condition may occur. If the "nature" of the error is "minor" the robot will self-repair and then continue. If the "nature" of the error is "fatal", the robot will terminate all activity immediately and indefinitely. （2）

Marks are awarded for using advanced notation that goes beyond using only states and transitions.

---------------------------------------------------------------------------------------



---------------------------------------------------------------------------------------

1. Briefly explain what superstates are typically used for and why they are considered to be an important feature of state diagrams. （3）[6 marks]

Complexity Reduction Through... Superstates » combination of several substates into one superstate » reduces complexity by hiding，aggregated states and multiplied transitions

A super-state is used when many transitions lead to the a certain state. Instead of showing all of the transitions from each state to the redundant state a super-state can be used to show that all of the states inside of the super-state can transition to the redundant state. This helps make the state diagram easier to read.

1. Name and very briefly describe three UML language/notation features designed to deal with complexity.

Complexity Reduction Through...

1. Superstates » combination of several substates into one superstate

» reduces complexity by hiding --- aggregated states and multiplied transitions

1. Concurrent Machines » parallel execution » reduces complexity by factorizing

* multiplied states and corresponding transitions

Create a UML state diagram that describes the states and events of a phone with the following behaviour:（3）

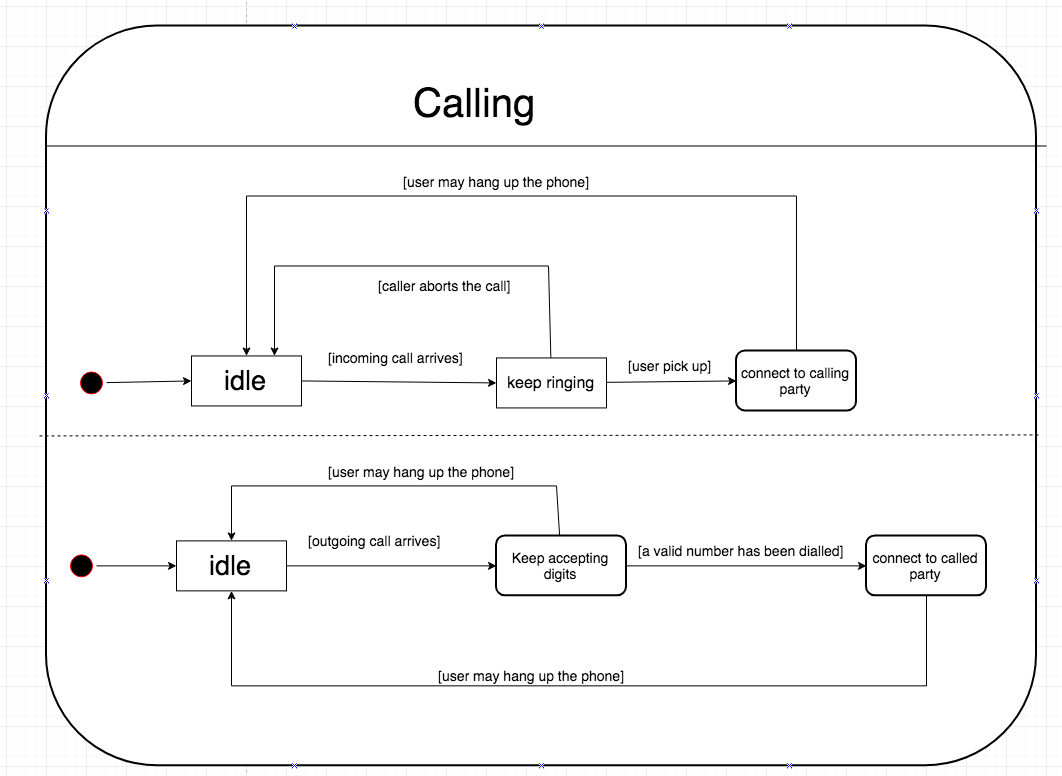
Initially, the phone is idle. When an incoming call arrives, it keeps ringing until the user picks up or the caller aborts the call. In the former case the phone is connected to the calling party, while in the latter case it becomes idle again.

For an outgoing call, when the user picks up the handle, the phone keeps accepting digits until a valid number has been dialled. When this happens, the phone becomes connected to the called party.

At any point during the dialling or while being connected the user may hang up the phone, causing it to become idle again.

Note: Marks are also awarded for the appropriate use of advanced notation.

---------------------------------------------------------------------

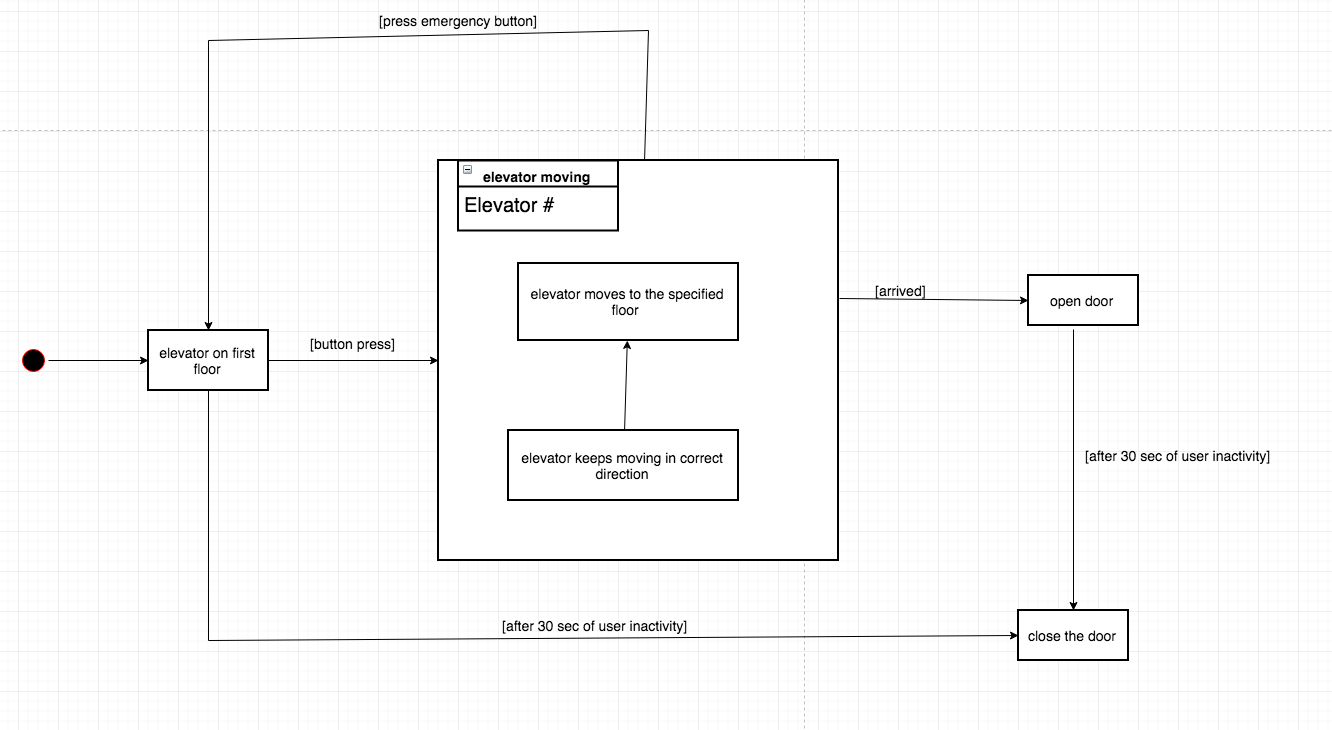


---------------------------------------------------------------------

(a) [15 marks] Create a UML state diagram that describes the behavior of an elevator. Initially, the elevator waits on the first floor. When a “button press” event occurs, the elevator moves to the floor number specified by the event. It is important for the elevator to move in the correct direction (up or down). When the elevator reaches the target floor, an “arrived” event is generated. The elevator should then open its doors. The elevator should close the door before it moves and return to the first floor after 30 seconds of user inactivity (i.e., no button presses). At any point during the elevator’s operation, it is possible to press the “emergency button” which will cause the elevator to return to the first floor.

Marks are awarded for the appropriate use of advanced notation.

--------------------------------------------------------------------------------



--------------------------------------------------------------------------------

(b) [5 marks] Briefly explain why substates, i.e., the ability to use concurrent lanes each specifying reactive behaviour that contributes to an overall combined behaviour, can be used to reduce the complexity of state diagrams.

Statecharts are OR diagrams » machine can be in multiple states

» extreme form are Petri Nets » the possible combinations constitute new global states themselves

different interpretation of a single substate

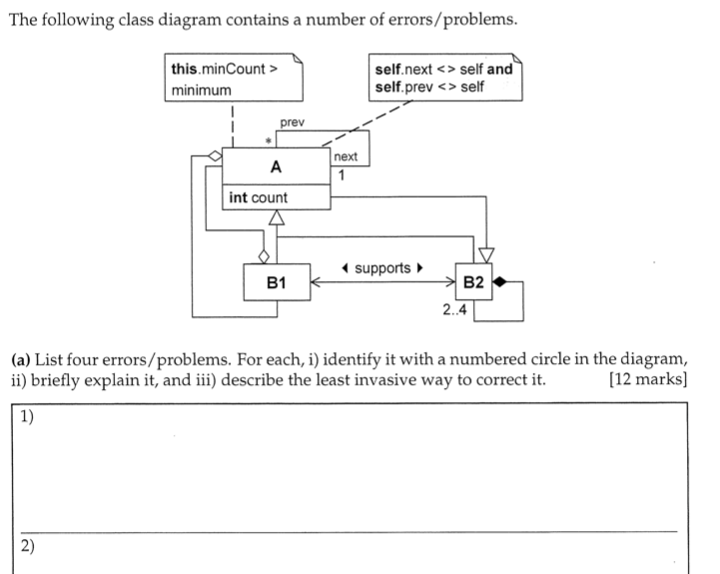
way to concisely describe large finite state machines

Concurrent Substates » parallel execution

» reduces complexity by factorising

multiplied states and corresponding transitions

Q5 Modelling

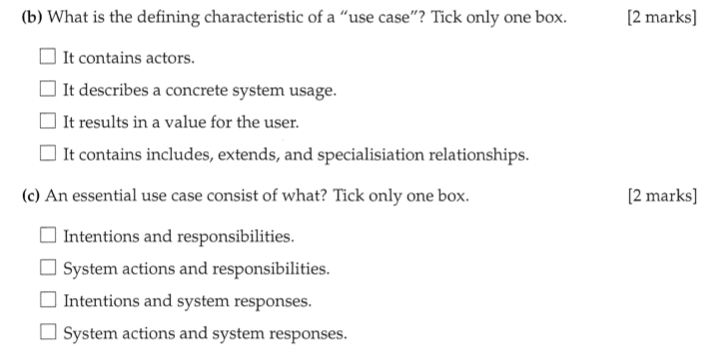


1)The attribute of class A should be changed into count: int because the correct format should be visibility - name of variable – type of variable

2)delete one of aggregation between class A and B1 because aggregation should be anti-symmetric and transitive

3)change multiplicities of the composition of B2 into one or more. because composition is used that parts cannot exist without the whole

4)delete the generalisation from A to B2. The reason is that if A is parent class of B2, therefore B2 can not be parent class of A



b-> it contains includes, extends and specialisation relationships.

c-> intentions and system responses

1. A colleague asks you what the direction of the inheritance relationship between the

concepts "Rectangle" and "Square" should be. Advise your colleague of three alternative options and briefly explain the rationale for each option. [8 marks] (3)

1. from rectangle to square

2. from square to rectangle

3. both rec and square inherit to another class, such as “shape”

Classification (is-a) » System understanding

Code Reuse (subclassing) » division between common and specialized code » easy library creation

Substitution principle (subtyping) » behavioural equality with extensions » easy library usage

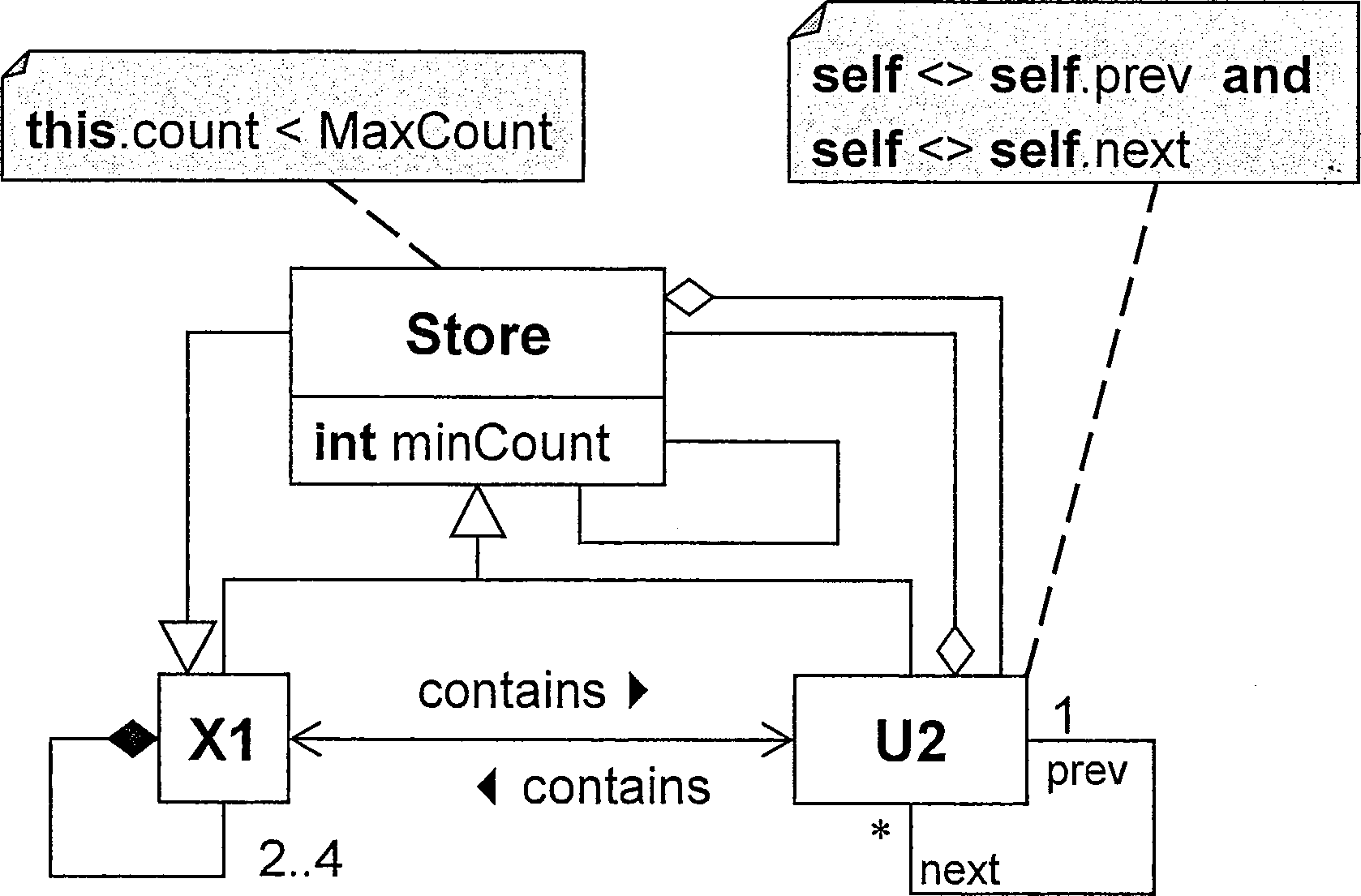
1. Briefly explain for which purpose you would choose a sequence diagram over a communication diagram and vice versa.

Sequence diagrams are for showing the order of system actions over time. I would choose a sequence diagram for showing the order of events and return sequences.

Communication diagrams are for showing the links between components in a system and how they communicate with each other. I would choose a communication diagram if I wanted to show a spatial perspective of the whole system.

b) Briefly explain when and why one would use substates, i.e., the ability to use concurrent lanes each specifying reactive behaviour that contributes to an overall combined behaviour.

The following class diagram contains a number of errors/problems.



(a) List four errors/problems. For each, i) identify it with a numbered circle in the diagram, ii) briefly explain it, and iii) describe the least invasive way to correct it.

1)The attribute of class A should be changed into count: int because the correct format should be visibility - name of variable – type of variable

2)delete one of aggregation between class A and B1 because aggregation should be anti-symmetric and transitive

3)change multiplicities of the composition of B2 into one or more. because composition is used that parts cannot exist without the whole

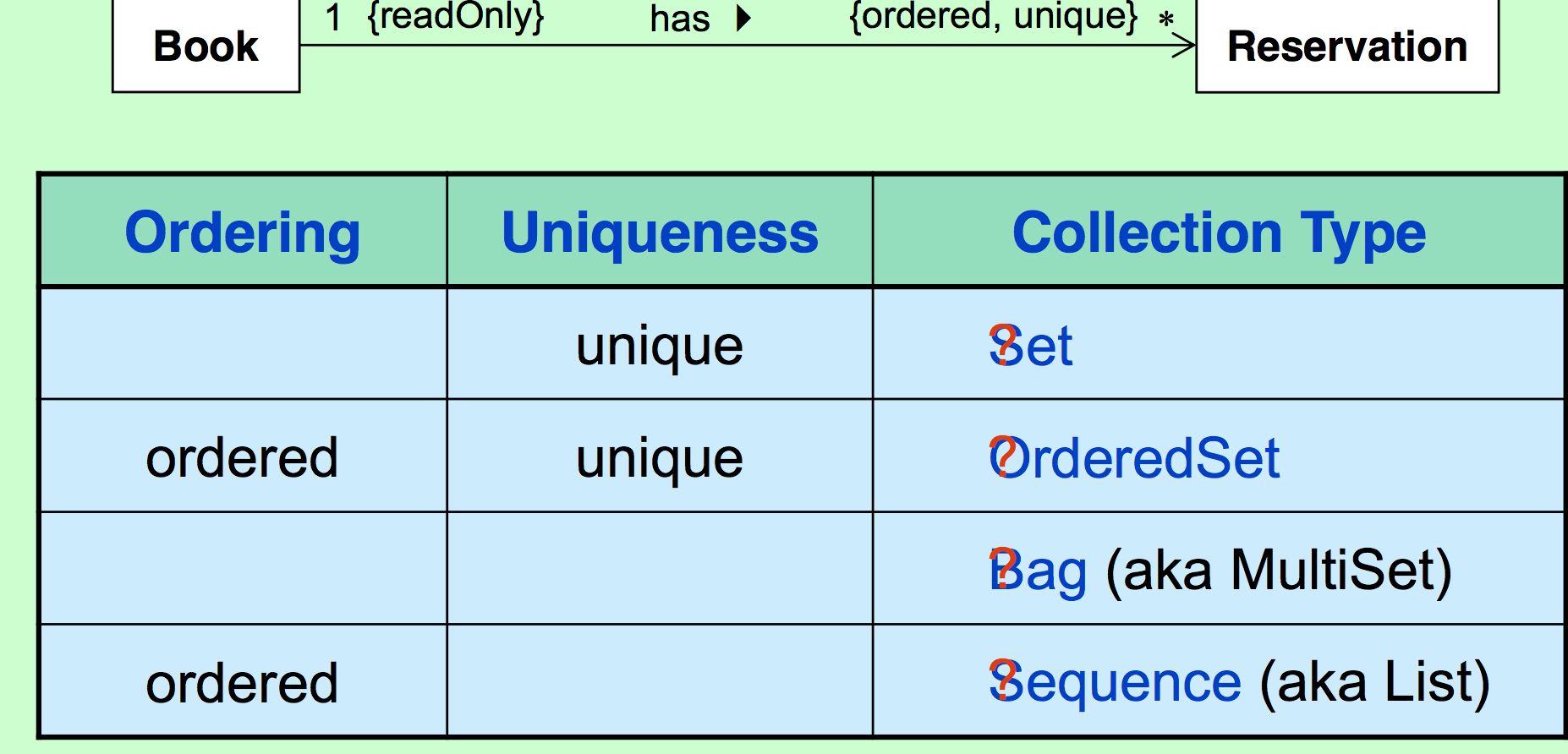
4)delete the generalisation from A to B2. The reason is that if A is parent class of B2, therefore B2 can not be parent class of A

­­ [4 marks] List and discuss the properties that make a good class diagram.

**Visibility of attribute and method in a class The permitted values**

**Multiplicity -> May have multiple associations between two classes if there are different relationships.**

Collection Types



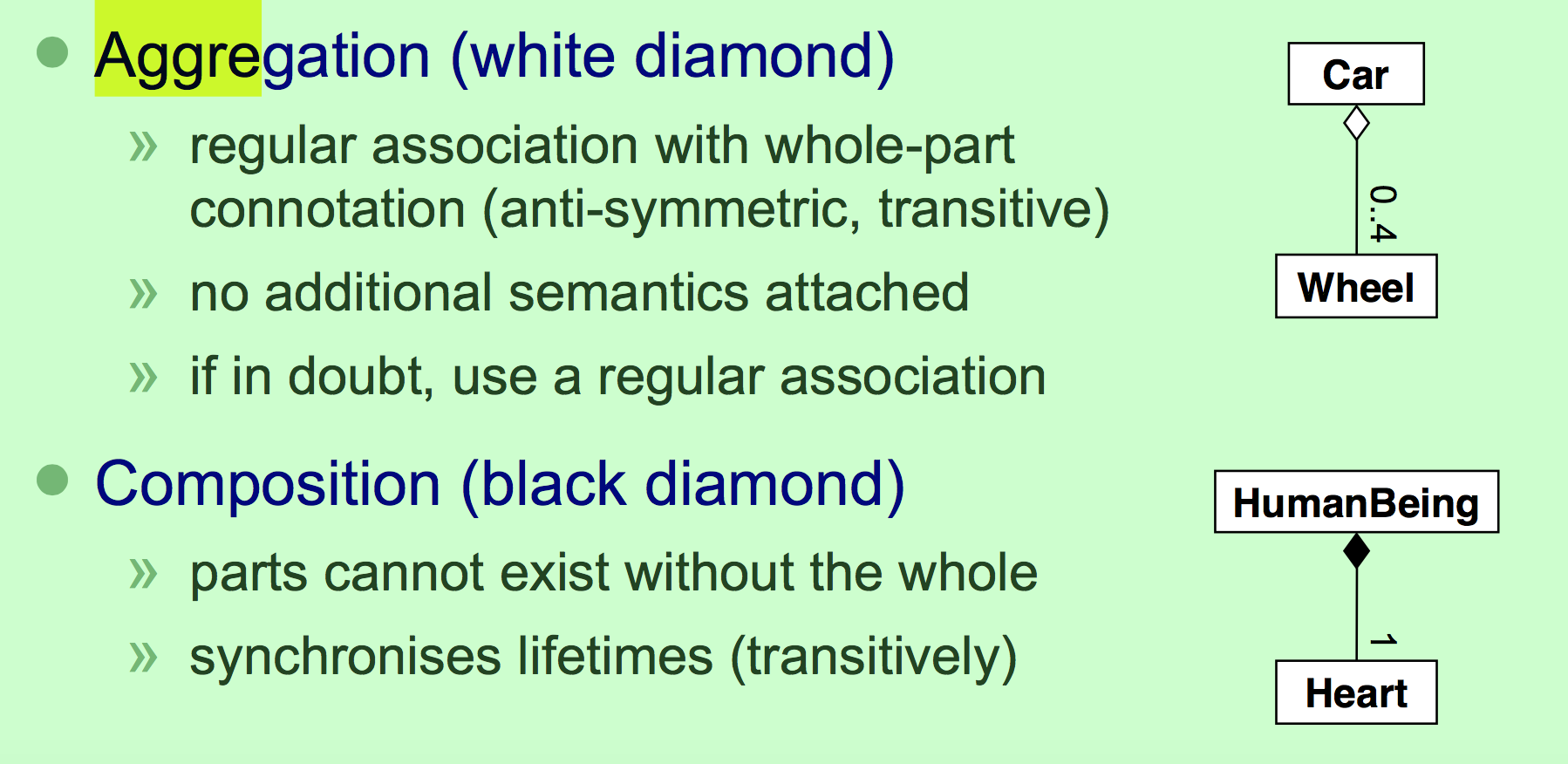
Generalisation is used for classification (System understanding), code reuse(subclassing) ---

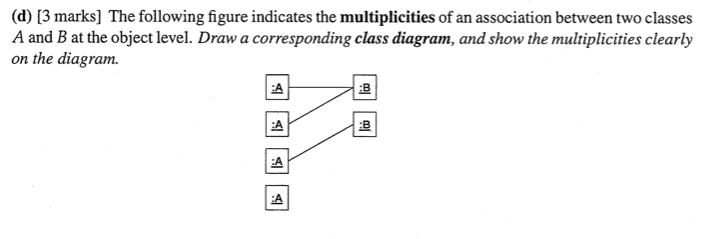
division between common and specialized code

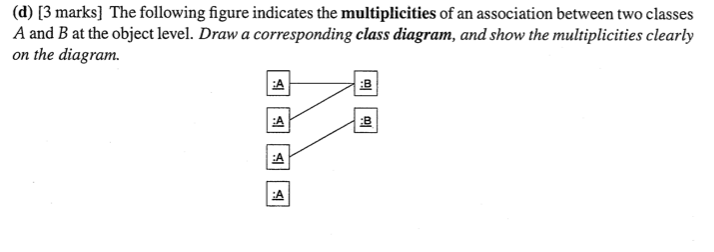
and substitution principle(subtyping) - » behavioural equality with extensions

[2 marks] Explain why class diagrams are already created in the analysis stage of software engineering.

[6 marks] Aggregation and composition are two kinds of association in class diagrams. Give an example for each of them. Discuss the differences between aggregation and composition. Use your examples to illustrate the differences.

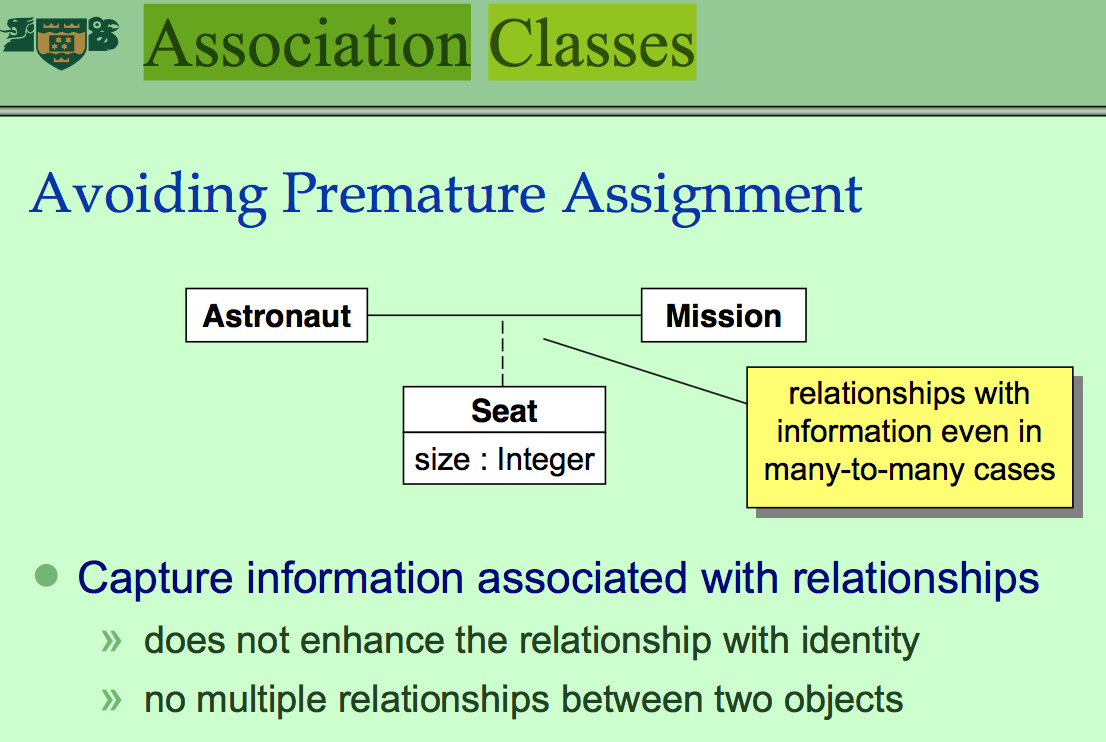






1..2 0…\*

1. [6 marks] Discuss why the use of association classes and higher-arity associations is a good idea. Give an example of a higher-arity association to illustrate the benefits.

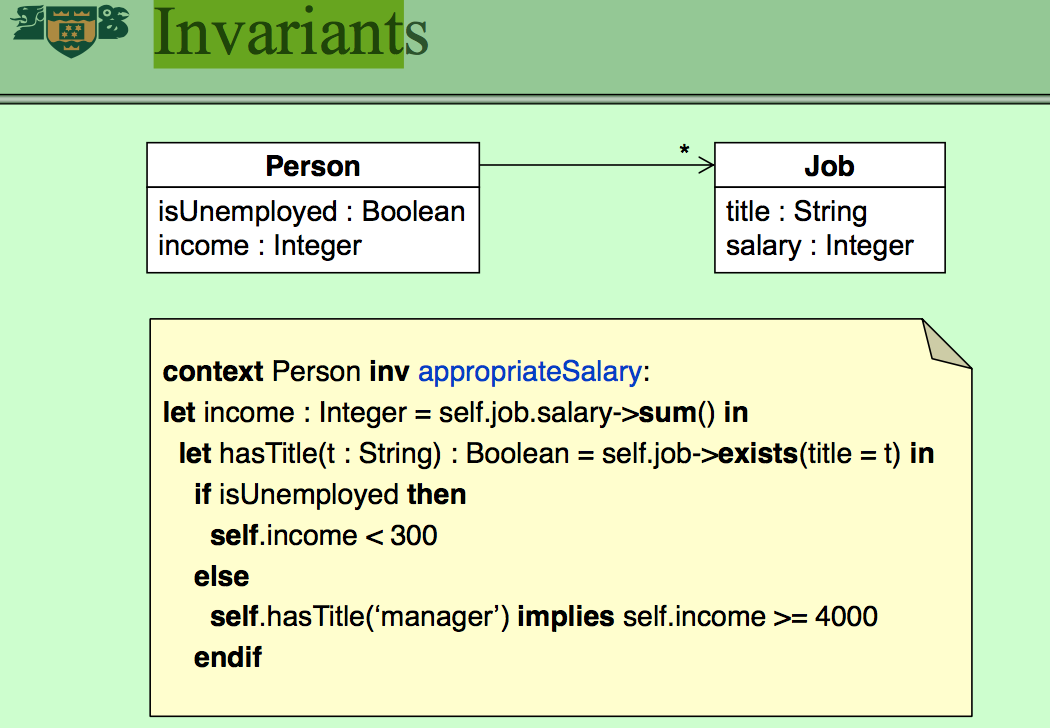


1. Explain the purpose of the Object Constraints Language (OCL).

OCL is frequently used to express model transformations (as part of the source and target patterns of transformation rules), well-formedness rules (as part of the definition of new domain-specific languages), or code-generation templates (as a way to express the generation patterns and rules)

1. [6 marks] What are invariants in the context of OCL? Give an example of an invariant using OCL and explain its meaning.

Invariant is one of types of Constraints. Invariant constraint that states a condition that must always be met by all instances of the type. Invariants must be true all the time (except during operation execution).



UML

(a)Briefly explain what a "use case" is. Include the ultimate criterion that determines whether something really should be regarded as a use case.

A use case is a detailed description of how a system is expected to interact with a user (or possibly another system).

Having such a description allows us to analyse it and develop specific:

functional requirements, actors,

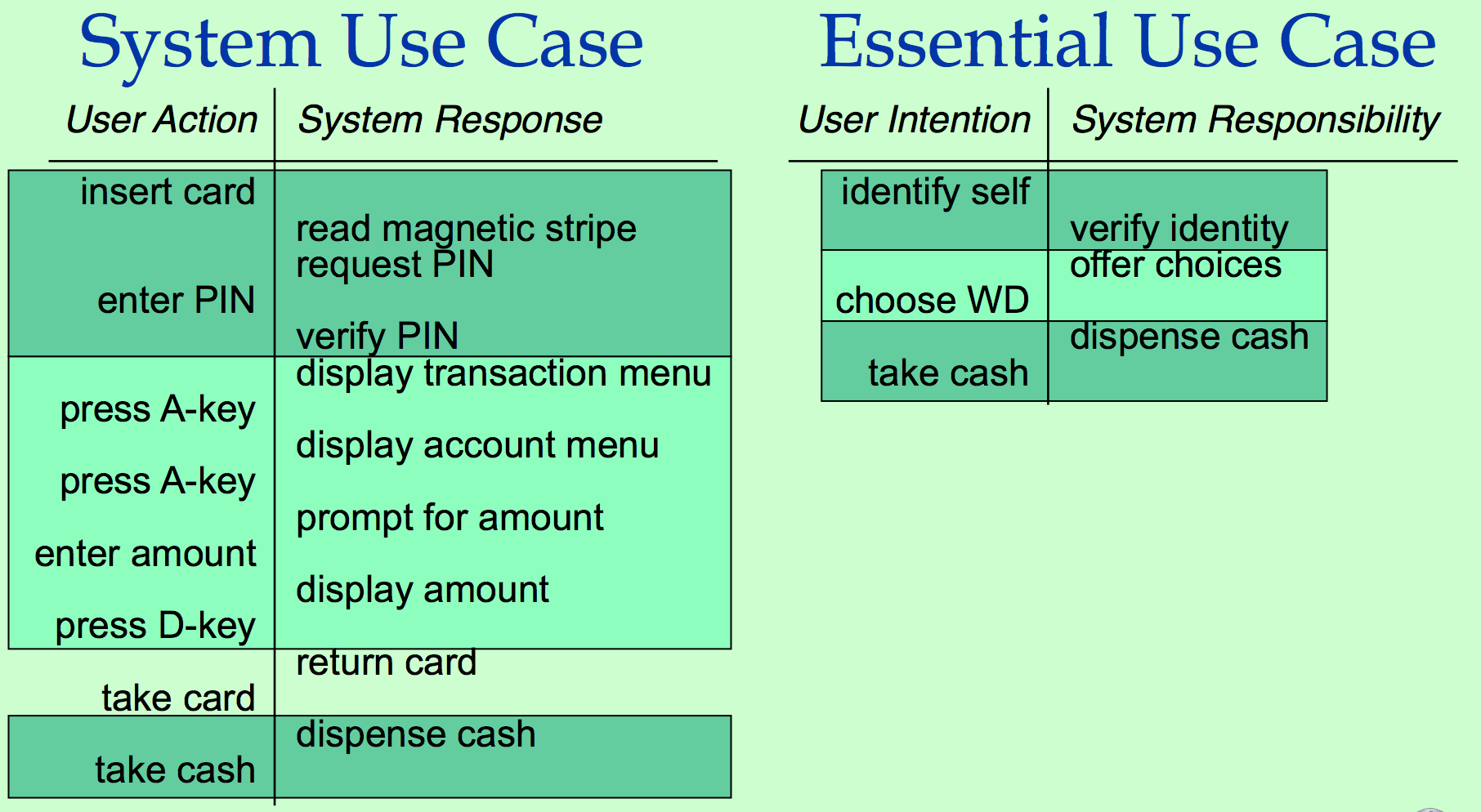
and interactions that the system must respond to in an appropriate manner.

(b)[2 marks] Briefly explain the idea of an "essential use case" (as opposed to a "system use case").

System Use Case » describes user actions and system responses at a technical level

• Essential Use Case » describes user intentions and system responsibilities

a simplified and generalized form of use case ... intended to capture the essence of problems through technology-free, idealized, and abstract descriptions

(b)Briefly explain the difference between a "system use case" and an "essential use case".

System Use Case » describes user actions and system responses at a technical level

• Essential Use Case » describes user intentions and system responsibilities

Describe two different kinds of situations when one would need to add textual constraints to a conceptual diagram. Provide an example for each respectively. [6 marks]

Consider a high security facility. Access is possibly only to individuals that can identify themselves with a staff card or per finger print reader. In rare "alert" situations, access is possible only for a subset of "VIP" users.

1. Sally wrote the following OCL constraint in order express that if an employed person is a manager, the person's income must be at least $4000:

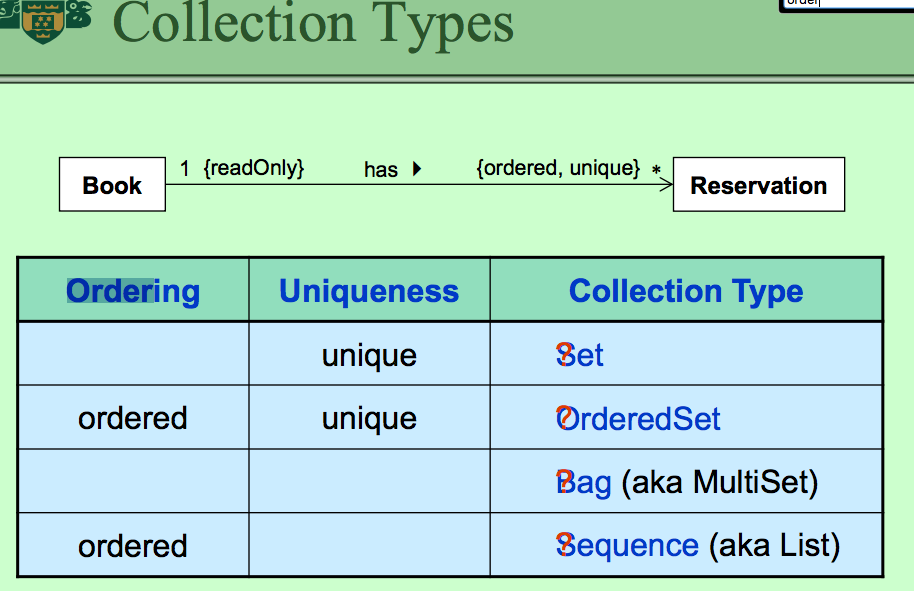
context Person inv appropriateSalary: if self.isEmployed then

self.isManager and self.income >= 4000 endif

What would you change and why?

Easier to read. The first could imply that any employee is automatically a manager whereas the second one implies that self.isManager is a conditional.

(b)Briefly describe the UML's approach to characterising container types and write down how you would specify that a concept is used as i) an "Ordered Set" and ii) a "Sequence".



(d)In what way can the potentially rich use case "Access Facility" be factored into multiple smaller parts that are easier to deal with on their own? Base your factorization on the three UML use case relationships.

(e)Draw the UML use case diagram for your design of question (d).

Describe two different kinds of situations when one would need OCL constraints in modelling. Provide an example for each respectively.

Consider a university library. When library users return items they need to identify them-selves through their library card or a staff card. In rare cases it may not be possible for the library staff to accept an item because the latter needs to be handled by a special staff member that is absent.

(d) [4 marks] In what way can the potentially rich use case "Return Item" be factored into multiple smaller parts that are easier to deal with on their own? Base your factorization on the three UML use case relationships.

(e)[5 marks] Draw the UML use case diagram for your design of question (d