CS/IT 413

Hall Ticket Number:

IV/IV B.Tech (Regular) DEGREE EXAMINATION

November, 2016 **Computer Science and Engineering**

Seventh Semester **Object Oriented Analysis And design**

Time: Three Hours **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1. Answer all questions
(1X12=12 Marks)

a What is an information system?
b Compare model and diagram?
c Define requirement?
d What are the three basic type of attributes?
e Define use case modelling?
f What is meant by system behaviour?
g Define pattern template?
h Define OCL?
i Define External event?
j Differentiate between state and event?
k Define Unified Process?
l Define legacy system.

UNIT – I

2.a What are the fact finding techniques for gathering requirements. Explain? 8M
2.b What the main advantages of object oriented development? 4M

(OR)

3.a Briefly explain use case relationships with suitable examples 6M
3.b Draw a use case diagram for Library management system? 6M

UNIT – II

4.a Draw a sequence diagram for student course registration system? 8M
4.b Differentiate between sequence diagram and collaboration diagram? 4M

(OR)

5.a Explain about state chart diagram with suitable examples? 4M
5.b Draw a state chart diagram for ATM system? 8M

UNIT – III

6.a Briefly discuss about different types of design. 6M
6.b Explain about the objectives of good design. 6M

(OR)

7.a Explain the criteria for good design. 6M
7.b Explain about singleton, structural and behavioral patterns. 6M

UNIT – IV

8.a Draw the component diagram for library management system? 6M
8.b Explain about DSDM and XP process models 6M

(OR)

9.a	Explain about prototyping the user interface.	6M
9.b	What is reuse? Explain the strategy planned for reuse.	6M

IV/IV B.Tech (Regular) DEGREE EXAMINATION

October, 2016 | **Common for CSE & IT**
Seventh Semester | **Object Oriented Analysis and Design**
Time: Three hours | **Maximum Marks: 60**

Answer question No.1 compulsorily. (1*12=12 Marks)
Answer one question from each unit (4*12=48 Marks)

1. Answer all questions

a. What is an information system?

An information system is a work system whose activities are devoted to capturing, transmitting, storing, retrieving, manipulating and displaying information.

(Or)

An information system must be to produce something that meets the needs of the people who will be using it.

b. Compare model and diagram?

MODEL: It is a simplification of reality. They are very useful in the following ways.

a. A model is a quicker and easier to build.
b. A model can be used in simulations to learn more about things in representation.
c. A model gives clear understanding of a problem.

Diagram:

Analysts and designers use diagrams to build models of systems in the same Way as architects use diagrams to model buildings.

a. To generate new ideas and possibilities.
b. To test ideas and more predictions.
c. To understand structures and relationships.

c. Define requirement.

It is important that information about what people are doing is gathered and documented which intern we call it as requirement.

d. What are the three basic types of attributes?

Boundary control and entity. ***(grace mark if attempted)***

e. Define use case modeling?

The use case model is called as the requirements model; which also include a problem domain object model and user interface descriptions in the requirements model.

f. What is meant by system behavior?

System behavior defines as what a system will do in response to its external environment without referring to details on implementation.

g. Define pattern template?

The *pattern template* determines the style and structure of the pattern description, and these vary in the emphasis they place on different aspects of patterns.

h. Define OCL.

OCL is used for the construction of pre-defined elements and *types,* and the language has a precise grammar that enables the construction of unambiguous statements about the properties of model components and their relationships to each other.

i. **Define External event.**

The transition of a state that is caused by any of the external activities is called as external event

j. **Differentiate between state and event.**

State:

All objects will have a state in a system. The current state of an object is a result of the events that have occurred to the object, and is determined by the current value of the object's attributes and the links that it has with other objects.

Event:

An event is an occurrence of a stimulus that can trigger a state change and that is relevant to the object or to an application

k. **Define Unified Process.**

Unified Process is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP)

l. **Define legacy systems.**

Legacy systems means any computerized information system that has been in use for some time, that was built with older technologies may be using a different development approach at different times and, most importantly, that continues to deliver benefit to the organization

UNIT-I

2. a) What are the fact finding techniques for gathering requirements? Explain. 8M

Fact finding techniques:

There are five main fact finding techniques that are used by analysts to investigate requirements.

1. Background Reading.
2. Interviewing.
3. Observation.
4. Document sampling.
5. Questionnaires

All fact finding techniques explanation ------5M
Advantages--------1M
Disadvantages----1M
Appropriate situation---1M

1 Background reading

If an analyst is assigned within the organization that is the subject of the fact gathering exercise, then he or she will already have a good understanding of the organization and its business objectives. If, however, he or she is going in as an outside consultant, then one of the first tasks is to try to gain an understanding of the organization. Background reading or research is part of that process.

The kind of documents that are suitable sources of information include the following:

a. company reports,
b. organization charts,
c. policy manuals,
d. job descriptions,
e. Reports and
f. Documentation of existing systems.

Reading company reports may provide the analyst with information about the organization's mission, and so possibly some indication of future requirements, this technique mainly provides information about the current system.

Advantages and disadvantages

+ Background reading helps the analyst to get an understanding of the organization before meeting the people who work there.

+ It also allows the analyst to prepare for other types of fact finding, for example, by being aware of the business objectives of the organization.

- Written documents often do not match up to reality; they may be out of date or they may reflect the official policy on matters which deals differently in practice.

Appropriate situations

- Background reading is appropriate for projects where the analyst is not familiar with the organization being investigated.
- It is useful in the initial stages of investigation.

2 Interviewing

Interviewing is probably the most widely used fact finding technique; it is also the one that requires the most skill and sensitivity.

Interviews can be used to gather information from management about their objectives for the organization and for the new information system, from staff about their existing jobs and their information needs, and from customers and members of the public as possible users of systems. While conducting an interview, the analyst can also use the opportunity to gather documents that the interviewee uses in his or her work.

Advantages and disadvantages

+ Personal contact allows the analyst to be responsive and adapt to what the user says. Because of this, interviews produce high quality information.

+ The analyst can investigate in greater depth about the person's work than can be achieved with other methods.

+ If the interviewee has nothing to say, the interview can be terminated.

- Interviews are time-consuming and can be the most costly form of fact gathering.

- Interview results require the analyst to work on them after the interview: the transcription of tape recordings or writing up of notes.

Appropriate situations

Interviews are appropriate in most projects. They can provide information in depth about the existing system and about people's requirements for a new system.

3 Observation

Watching people carrying out their work in a natural setting can provide the analyst with a better understanding of the job than interviews, in which the interviewee will often concentrate on the normal aspects of the job and forget the exceptional situations and interruptions which can occur with the system and how to cope up with those problems.

Observation also allows the analyst to see what information people use to carry out their job. This can tell you about the documents they refer to, whether they have to get up from their desks to get information, how well the existing system handles their needs.

Observation can be an open-ended process in which the analyst simply sets out to observe what happens and to note it down, or it can be a closed process in which the analyst wishes to observe specific aspects of the job and draws up an observation schedule or form on which to record data.

Advantages and disadvantages

+ Observation of people at work provides first hand experience of the way that the current system operates.

+ Data are collected in real time and can have a high level of validity if care is taken in how the technique is used.

+ Observation can be used to verify information from other sources or to look for exceptions to

the standard procedure.

+ Baseline data about the performance of the existing system and of users can be collected.

- Most people do not like being observed and are likely to behave differently from the way in which they would normally behave. This can distort findings and affect the validity.
- Observation requires a trained and skilled observer for it to be most effective.

Appropriate situations

- Observation is essential for gathering quantitative data about people's jobs.
- It can verify or disprove assertions made by interviewees, and is often useful in situations where different interviewees have provided conflicting information about the way the system works.

Document sampling

Document sampling can be used in two different ways.

First, the analyst will collect copies of blank and completed documents during the course of interviews and observation sessions. These will be used to determine the information that is used by people in their work, and the inputs to and outputs from processes which they carry out, either manually or using an existing computer system. From an existing system, the analyst may need to collect screen shots in order to understand the inputs and outputs of the existing system.

Second, the analyst may carry out a statistical analysis of documents in order to find out about patterns of data. For example, many documents such as order forms contain a header section and a number of lines of detail. The analyst may want to know the distribution of the number of lines in an order.

Advantages and disadvantages

+ Can be used to gather quantitative data, such as the average number of lines on an invoice.
+ Can be used to find out about error rates in paper documents.

- If the system is going to change dramatically, existing documents may not reflect how it will be in future.

Appropriate situations

The first type of document sampling is almost always appropriate. Paper-based documents give a good idea of what is happening in the current system. They also provide supporting evidence for the information gathered from interviews or observation.

5 Questionnaires

Questionnaires are a research instrument that can be applied to fact finding in system development projects. They consist of a series of written questions. The questionnaire designer usually limits the range of replies that respondents can make by giving them a choice of options.

YES/NO questions only give the respondent two options. If there are more options, the multiple choice type of question is often used when the answer is factual, whereas scaled questions are used if the answer involves an element of subjectivity. Some questions do not have a fixed number of responses, and must be left open-ended for the respondent to enter what they like. Where the respondent has a limited number of choices, these are usually coded with a number, which speeds up data entry if the responses are to be analysed by computer software. If you plan to use questionnaires for requirements gathering, they need very careful design.

Advantages and disadvantages

+ An economical way of gathering data from a large number of people.
+ If the questionnaire is well designed, then the results can be analyzed easily.

- Good questionnaires are difficult to construct.
- There is no automatic mechanism for follow up or probing more deeply, although it is possible to follow up with an interview by telephone or in person if necessary.
- Postal questionnaires suffer from low response rates.

2. b) What are the main advantages of object oriented development? 4M

Object Oriented Development (OOD) has been touted as the next great advance in software engineering. It promises to reduce development time, reduce the time and resources required to maintain existing applications, increase code reuse, and provide a competitive advantage to organizations that use it. While the potential benefits and advantages of OOD are real, excessive hype has lead to unrealistic expectations among executives and managers. Even software developers often miss the subtle but profound differences between OOD and classic software development.

Many benefits are cited for OOD, often to an unrealistic degree. Some of these potential benefits are

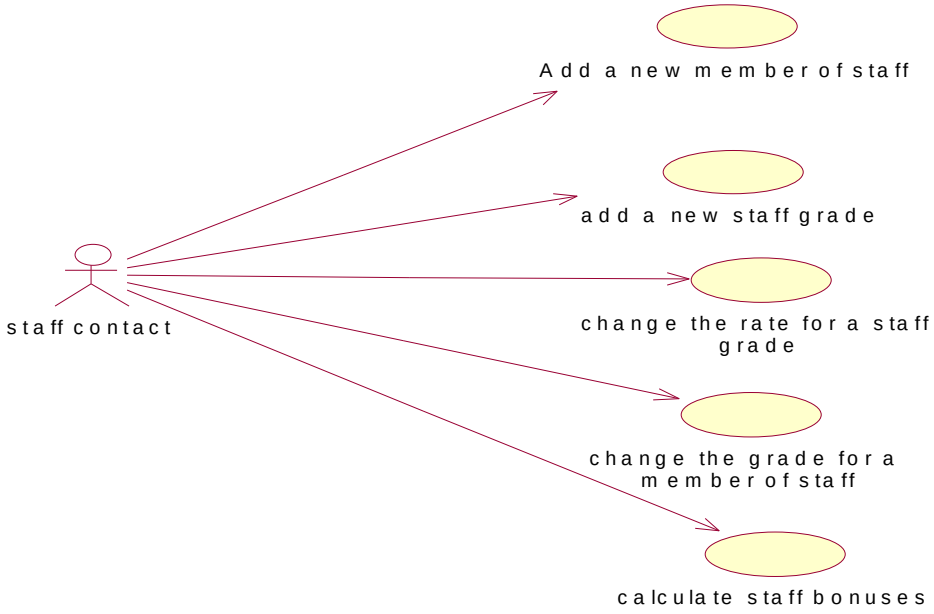
- *Faster Development:* OOD has long been touted as leading to faster development. Many of the claims of potentially reduced development time are correct in principle, if a bit overstated.
- *Reuse of Previous work:* This is the benefit cited most commonly in literature, particularly in business periodicals. OOD produces software modules that can be plugged into one another, which allows creation of new programs. However, such reuse does not come easily. It takes planning and investment.
- *Increased Quality:* Increases in quality are largely a by-product of this program reuse. If 90% of a new application consists of proven, existing components, then only the remaining 10% of the code has to be tested from scratch. That observation implies an order-of-magnitude reduction in defects.
- *Modular Architecture:* Object-oriented systems have a natural structure for modular design: objects, subsystems, framework, and so on. Thus, OOD systems are easier to modify. OOD systems can be altered in fundamental ways without ever breaking up since changes are neatly encapsulated. However, nothing in OOD guarantees or requires that the code produced will be modular. The same level of care in design and implementation is required to produce a modular structure in OOD, as it is for any form of software development.
- *Client/Server Applications:* By their very nature, client/server applications involve transmission of messages back and forth over a network, and the object-message paradigm of OOD meshes well with the physical and conceptual architecture of client/server applications.
- *Better Mapping to the Problem Domain:* This is a clear winner for OOD, particularly when the project maps to the real world. Whether objects represent customers, machinery, banks, sensors or pieces of paper, they can provide a clean, self-contained implication which fits naturally into human thought processes.

3. a) Briefly explain use case relationships with suitable examples. 6M

Use cases are descriptions of the functionality of the system from the users' perspective.

Use case diagrams are used to show the functionality that the system will provide and to show which users will communicate with the system in some way to use that functionality. The following figure shows an example of a use case diagram.

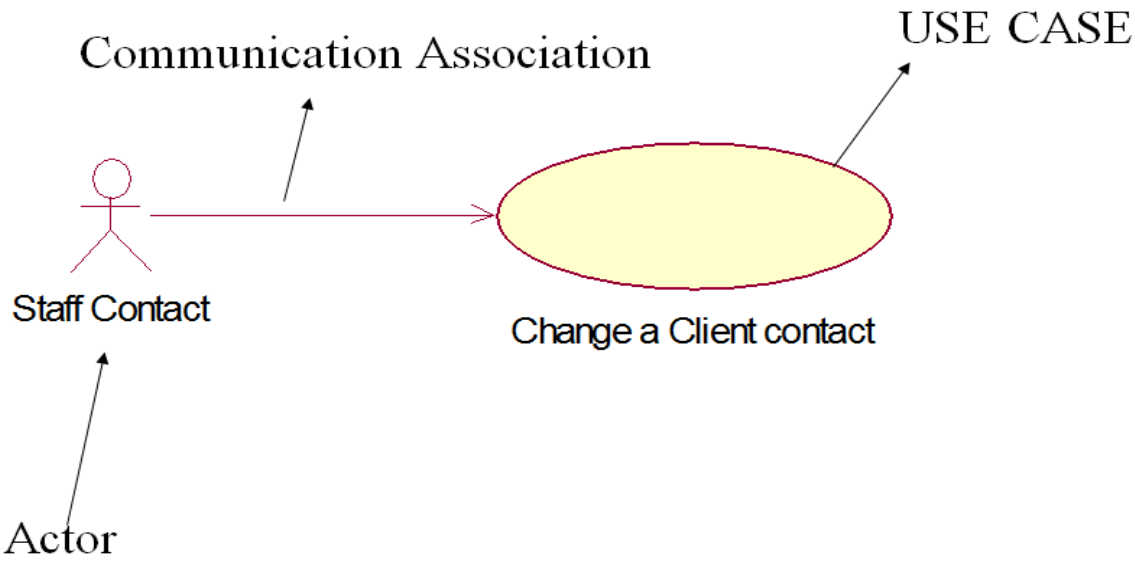
Use case relationship explanation—3M
Examples --------3M



Notation of USE CASE Diagrams

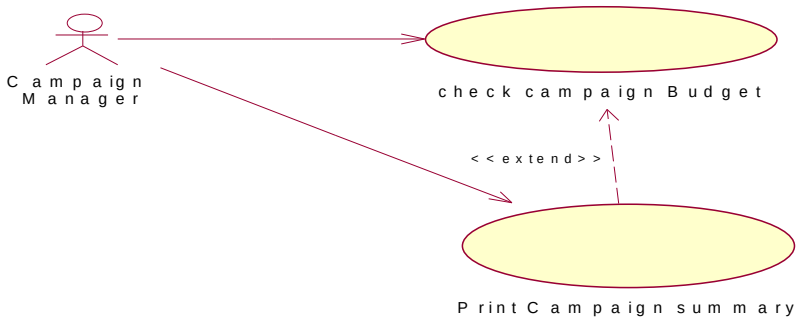
Use case diagrams show three aspects of the system: actors, use cases and relationships of the system or sub-system boundary.

The following Figure shows the elements of the notation.



Two further kinds of relationships can be shown on the use case diagram itself. These are the *Extend* and *Include* relationships.

1. «extend» is used when you wish to show that a use case provides additional functionality that may be required in another use case. In the following Figure , the use case Print campaign summary extends Check campaign budget.

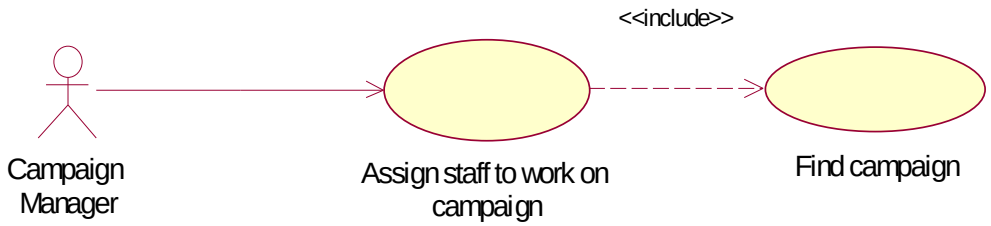


This means that at a particular point in Check Campaign Budget the user can optionally invoke the behaviour of Print campaign summary, which does something over and above what is done in Check campaign budget.

There may be more than one way of extending a particular use case, and these possibilities may represent significant variations on the way the user uses the system. Rather than trying to capture all these variations in one use case, you would document the core functionality in one and then extend it in others.

2. «include» applies when there is a sequence of behaviour that is used frequently in a number of use cases, and you want to avoid copying the same description of it into each use case in which it is used. The following Figure shows that the use case Assign staff to work on a campaign has an «include» relationship with Find campaign.

   This means that when an actor uses Assign staff to work on a campaign the behaviour of Find campaign will also be included in order to select the relevant Campaign.

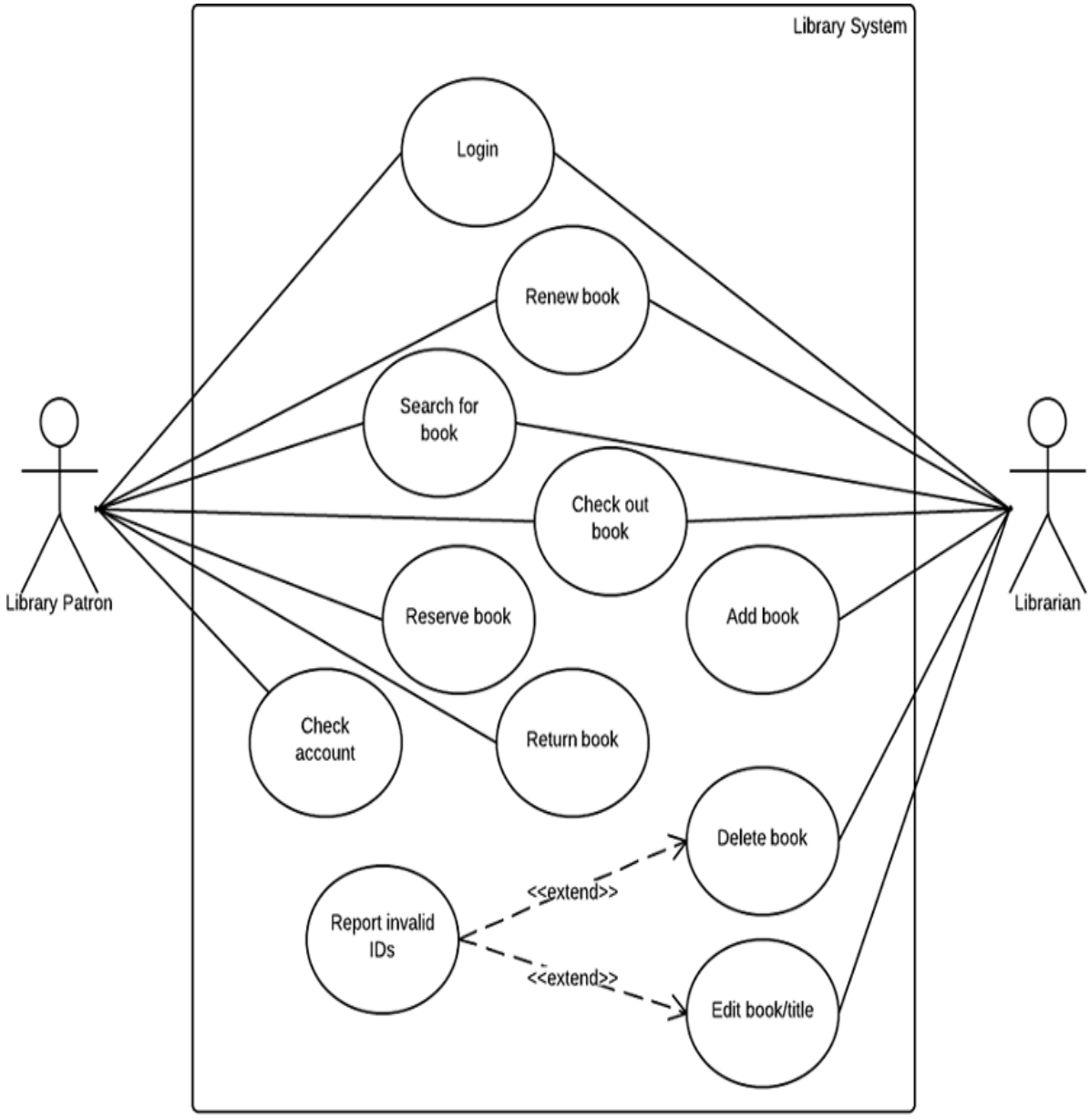


Its clear that actors need not be human users of the system. They can also be other systems that communicate with the one that is the subject of the systems development project, for example, other computers or automated machinery or equipment.

3. b) Draw a use case diagram for library management system? 6M

Any relevant example can be considered

Example diagram:



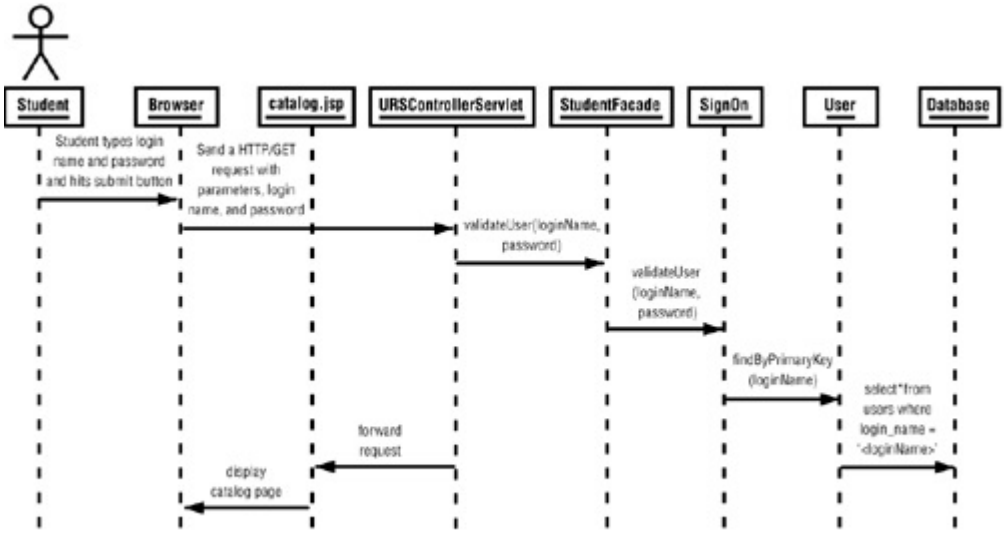
UNIT-II

4. a) Draw a sequence diagram for student course registration system? 8M

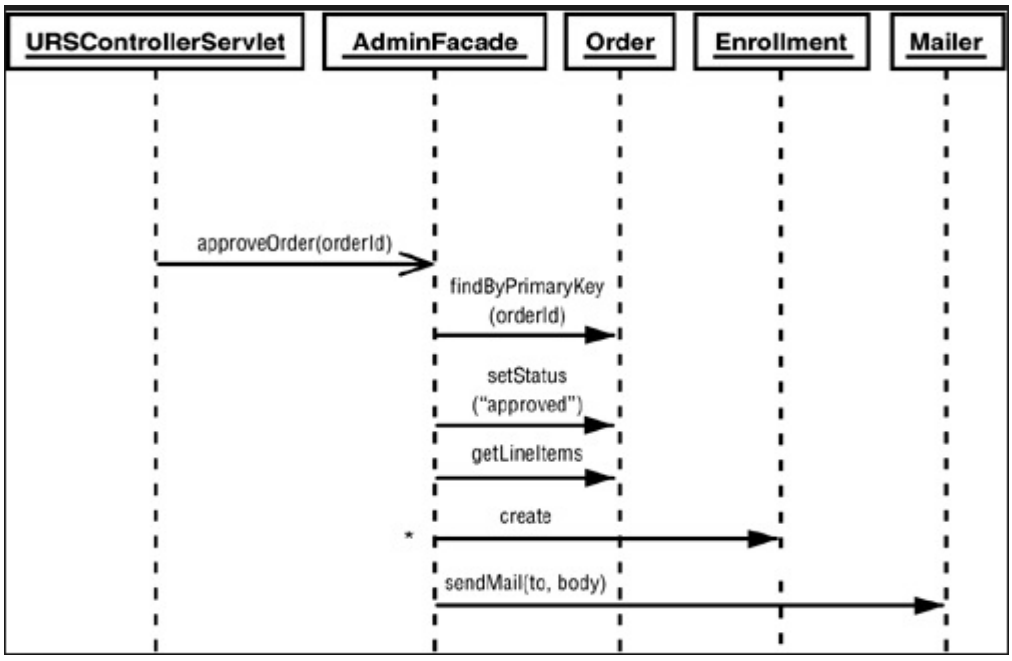
Example sequence diagram:

Any relevant example can be considered

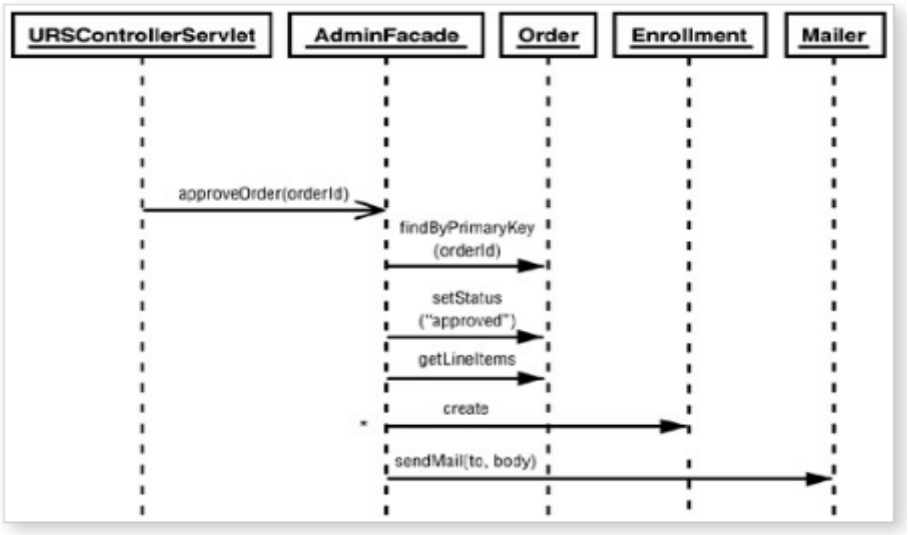
STUDENT LOGS ON TO THE SYSTEM



STUDENT PLACES ORDER SEQUENCE DIAGRAM.



ADMINISTRATOR APPROVES ORDER AND ENROLLS STUDENT SEQUENCE DIAGRAM.



4. b) Differentiate between sequence and collaboration diagram. 4M

Collaboration diagrams:

- Collaboration diagram illustrates object interactions in a graph or network format, in which objects can be placed anywhere on the diagram. It demonstrates how objects are statically connected.
- A collaboration diagram shows the relationship between objects and the order of messages passed between them.
- The objects are listed as icons and arrows indicate the messages being passed between them.

Sequence diagrams:

- Sequence diagram illustrates interaction in a kind of fence format, in which each new object is added to the right. It generally shows the sequence of events that occur.
- A sequence diagram shows relations between objects.
- It should be read from left to right and from top to bottom.
- At the top of the diagram are names of objects that interact with each other. These are the concepts in the conceptual model.

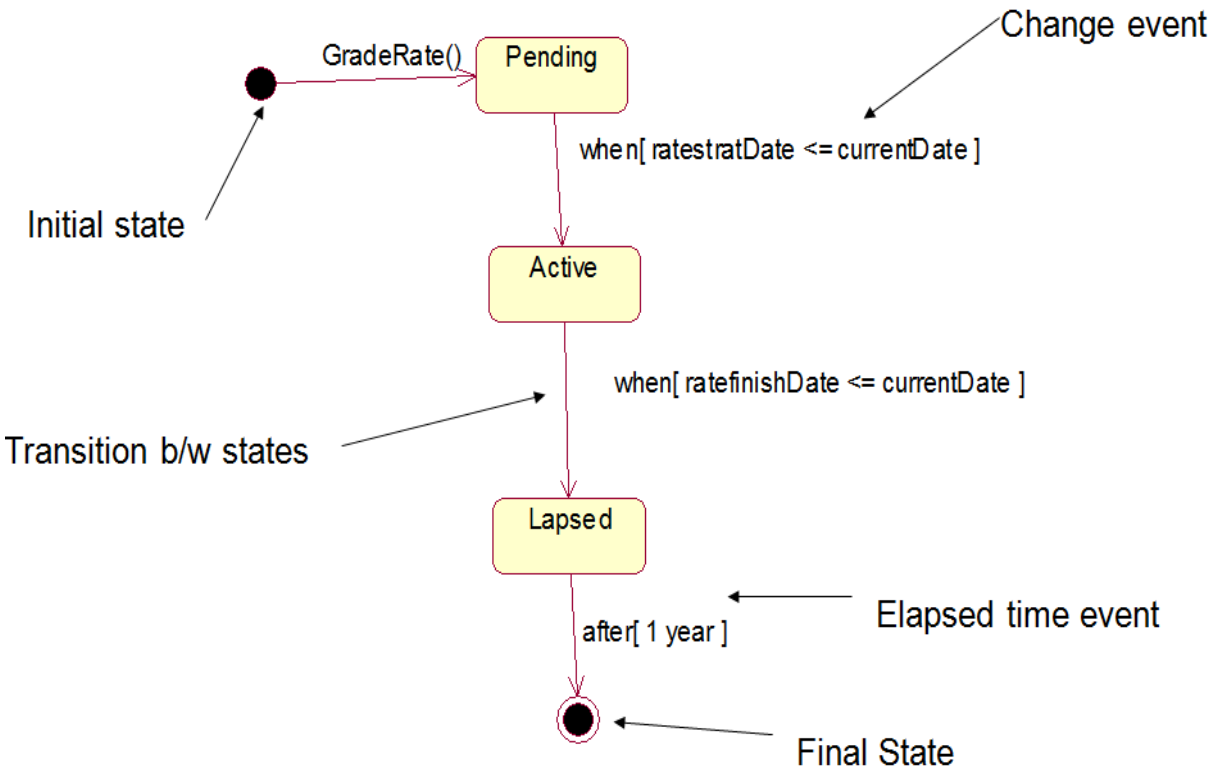
5. a) Explain about state chart diagram with suitable examples? **4M**

State chart explanation-------2M

Example----------2M

The statechart is a versatile technique, and can be used within an object-oriented approach for other purposes than the modelling of object life cycles. A statechart diagram shows the states of a single object, the events or messages that cause a transition from one state to another , and the actions that result from a state change. As in Activity diagram , statechart diagram also contains special symbols for start state and stop state.

It is important that movement from one state to another for a GradeRate object is dependent upon events that occur with the passage of time. The following figure shows a state chart for GradeRate.



A *call event* occurs when an object receives a call for one of its operations either from another object or from itself. Call events correspond to the receipt of a call message and are annotated by the signature of the operation as the trigger for the transition.

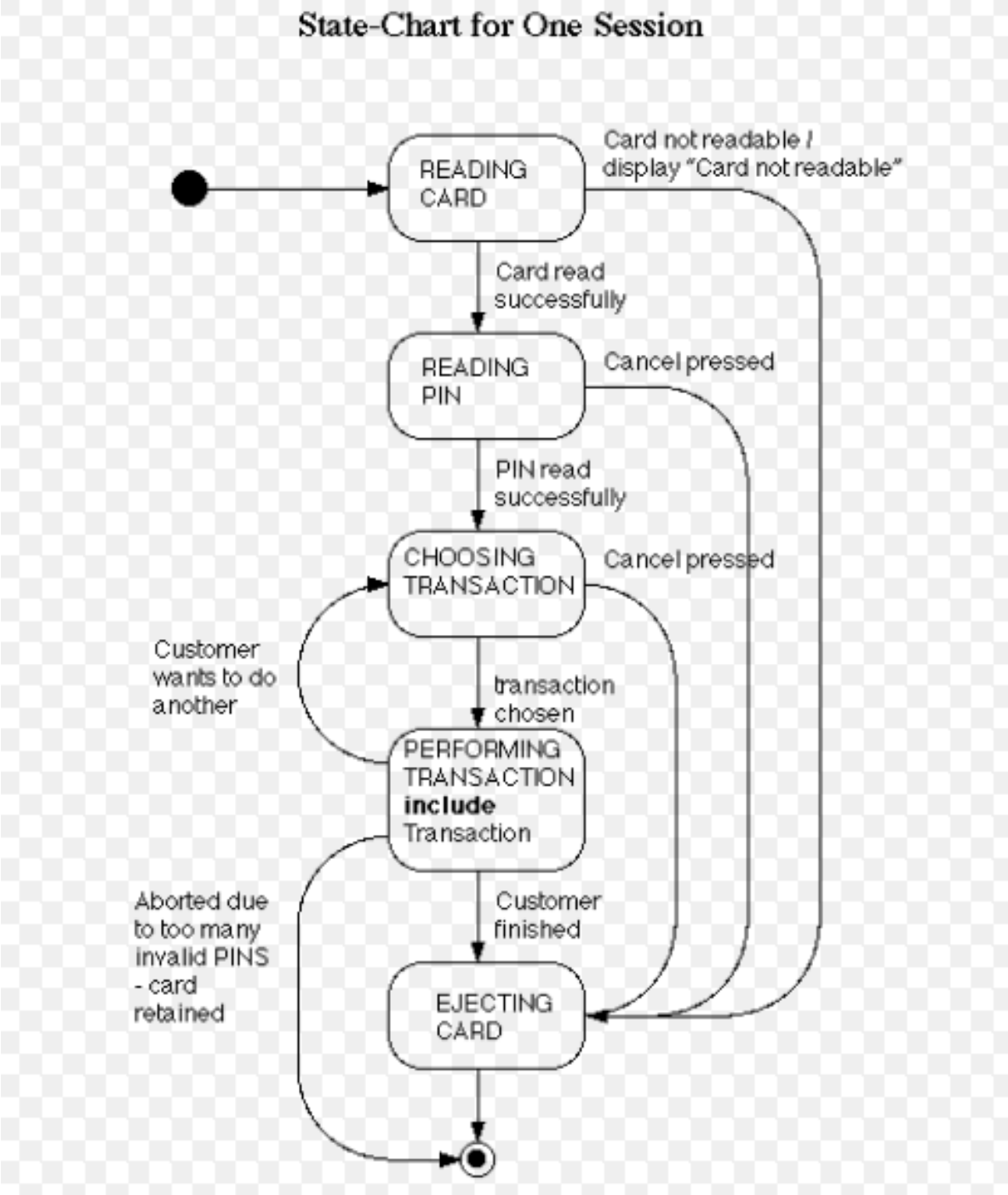
A *signal event* occurs when an object receives a signal. As with call events the event is annotated with the signature of the operation invoked. There is no syntactic difference between call events and signal events. It is assumed that a naming convention is used to distinguish between them.

An *elapsed-time event* is caused by the passage of a designated period of time after a specified event.

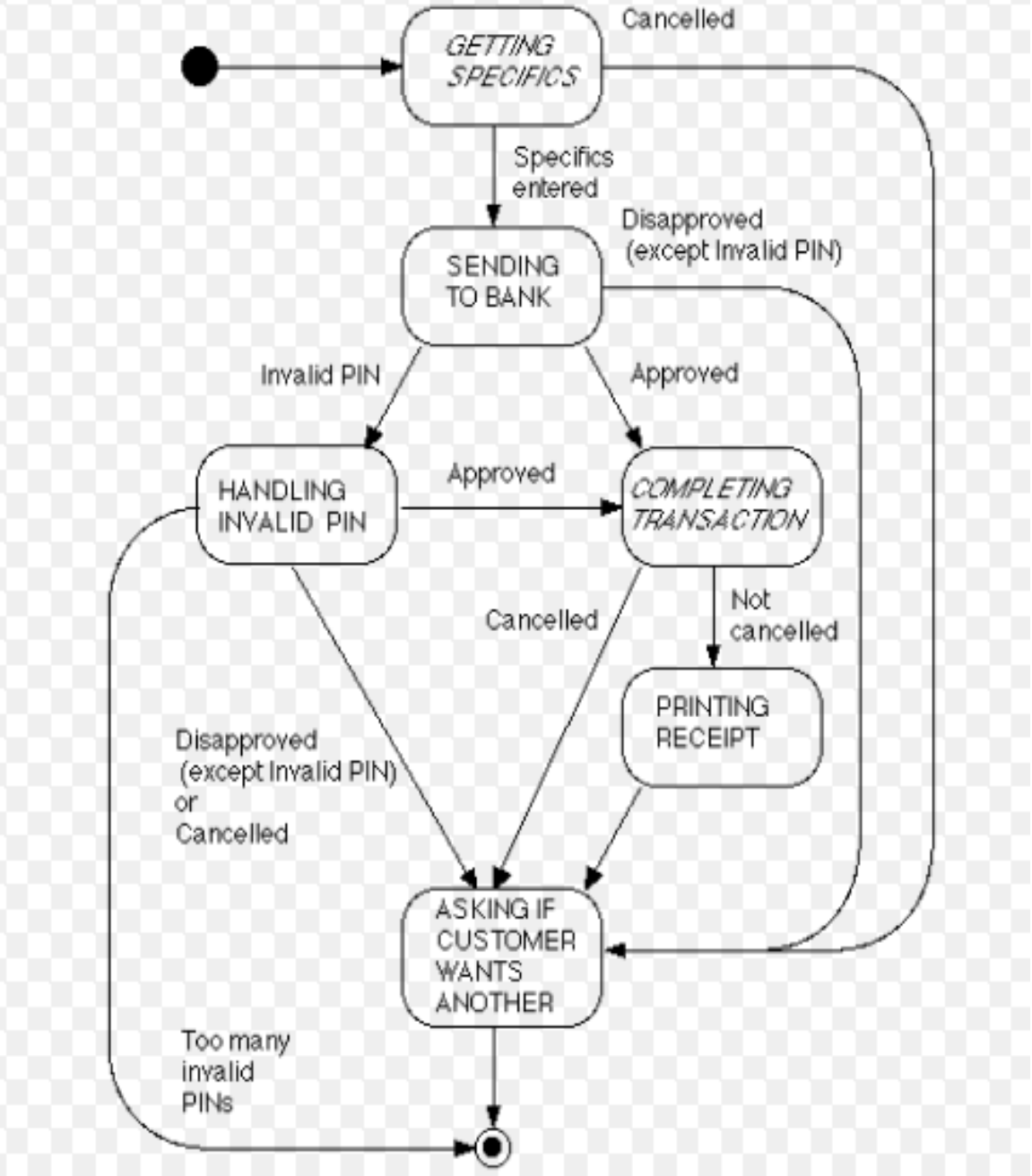
5. b) Draw a state chart diagram for ATM system? **8M**

Any relevant example can be considered

Example



State-Chart for One Transaction
(italicized operations are unique to each particular type of transaction)



UNIT-III

6. a) Briefly discuss about different types of design. 6M

Logical and physical design—2M
System design----2M
Detailed design-----2M

Logical design and Physical design

In the life of a system development project a decision must be made about the hardware and software that are to be used to develop and deliver the system-the hardware and software platform. In some projects this is known right from the start. Many companies have an existing investment in hardware and software, and any new project must use existing system software (such as programming languages and database management systems) and will be expected to run on the same hardware. This is more often the case in large companies with mainframe computers. In such companies the choice of configuration has been limited in the past to the use of terminals connected to the mainframe.

However, client-server architectures and open system standards, that allow for different hardware and software to operate together, have meant that even for such companies, the choice of platform is more open. For many new projects the choice of platform is relatively unconstrained, and so at some point in the life of the project a decision must be made about the platform to be used.

System Design and Detailed Design:

System design -

During system design the designers make decisions that will affect the system as a whole. The most important aspect of this is the overall architecture of the system. Many modern systems use a client-server architecture in which the work of the system is divided between the clients and a server. This arises questions about how processes and objects will be distributed on different machines, and it is the role of the system designer or system architect to decide on this.

Detailed Design –

This addresses the design of classes and the detailed working of system. These activities we need to consider under traditional and OO approaches

a) Traditional detailed design -

Here, detailed design was seen as consisting of four main activities:

- Designing inputs - Designing inputs meant designing the layout of menus and data entry screens;
- Designing outputs - Designing outputs concerned with the layout of enquiry screens, reports and printed documents;

- Designing processes - Designing processes dealt with the choice of algorithms and ensuring that processes correctly reflected the decisions that the software needed to make;

- Designing files - Designing files dealt with the structure of files and records, the file organization and the access methods used to update and retrieve data from the files.

b) Object-Oriented Detailed Design-

During the analysis phase of a project, concepts in the business will have been identified and elaborated in terms of classes, and use cases will have been identified and described. The classes that have been included in the class diagram will reflect the business requirements but they will only include a very simple view of the classes to handle the interface with the user, the interface with other systems, the storage of data and the overall co-ordination of the other classes into programs. These classes will be added in design with greater or lesser degrees of detail depending on the hardware and software platform that is being used for the new system.

6. b) Explain about the objectives of good design. 6M

Objectives of Design

The designers of a system seek to achieve many objectives that have been identified as the characteristics of a good design since the early days of information systems development.

Characteristics of Good Design are efficiency, flexibility, generality, maintainability and reliability and Other characteristics of a good design includes , it should be functional, portable, secure and economical in the context of object - oriented systems, reusability is a priority objective.

Functional. When we use a computer system, we expect it to perform correctly and completely those functions that it is claimed to perform; when an information system is developed for an organization, the staff of that organization will expect it to meet their documented requirements fully and according to specification.

Efficient. It is not enough that a system performs the required functionality; it should also do so efficiently, in terms both of time and resources. Those resources can include disk storage, processor time and network capacity. This is why design is not just about producing any solution, but about producing the best solution.

Economical , Linked to efficiency is the idea that a design should be economical.This applies not only to the fixed costs of the hardware and software that will be required to run it, but also to the running costs of the system.

Reliable. The system must be reliable in two ways: first, it should not be prone to either hardware or software failure; second it should reliably maintain the integrity of the data in the system.

Secure. Systems should be designed to be secure against malicious attack by outsiders and against unauthorized use by insiders. System design should include

considerations of how people are authorized to use the system and policies on passwords.

Flexible. It is the ability to adapt changing business requirements as time passes.

Generality. It Describes the extent to which a is general purpose.Means it includes the feature of portability.

Manageable. A good design should allow the project manager to estimate the amount of work involved in implementing the various sub-systems.If implementation of these sub-systems is completed then forwarded for testing will not have any affects on other parts of the system.

7. a) Explain the criteria for good design. 6M

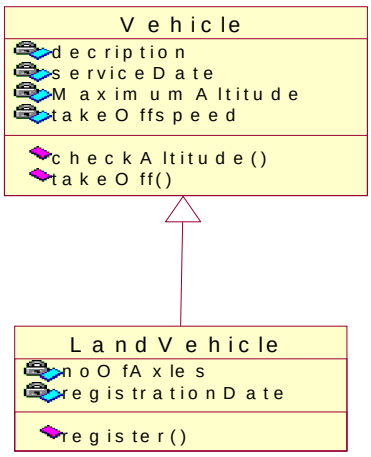
Criteria for Good Design

Coupling and cohesion – These factors coupling and cohesion are inportant factors for good design.

Coupling describes the degree of interconnectedness between design components and is reflected by the number of links an object has and by the degree of interaction the object has with other objects.

Cohesion is a measure of the degree to which an element contributes to a single purpose. The concepts of coupling and cohesion are not mutually exclusive but actually support each other. This criteria can be used within object-orientation as described below.

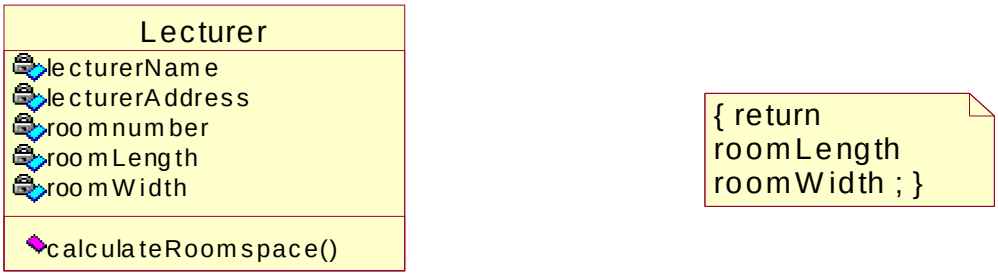
Interaction Coupling is a measure of the number of message types an object sends to other objects and the number of parameters passed with these message types. Interaction coupling should be kept to a minimum to reduce the possibility of changes rippling through the interfaces and to make reuse easier. When an object is reused in another application it will still need to send these messages and hence needs objects in the new application that provide these services.



Inheritance Coupling describes the degree to which a subclass actually needs the features it inherits from its base class.

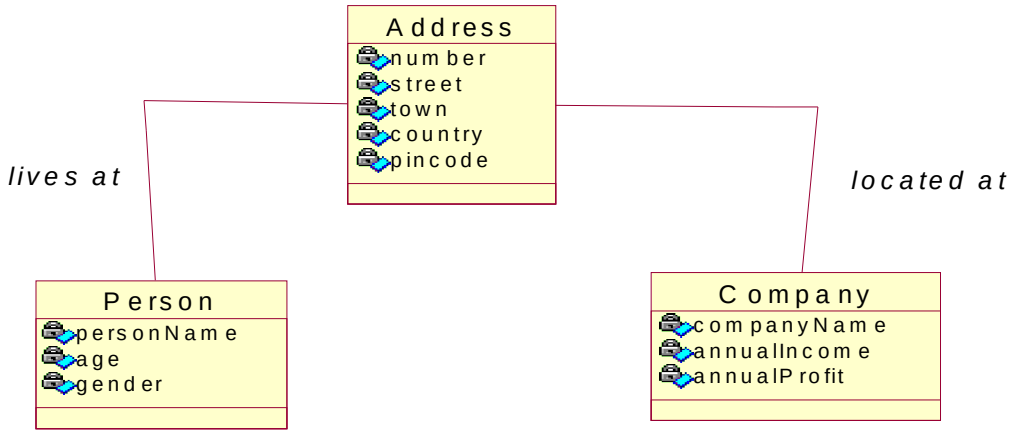
For example, in the above figure, the inheritance hierarchy exhibits low inheritance coupling and is poorly designed. The subclass LandVehicle needs neither the attributes maximumAltitude and takeOff Speed nor the operations checkAltitude () and takeOff (). They have been inherited unnecessarily.

Operation Cohesion measures the degree to which an operation focuses on a single functional requirement. Good design produces highly cohesive operations, each of which deals with a single functional requirement. For example in the following figure , the operation calculateRoomSpace () is highly cohesive.



Class Cohesion reflects the degree to which a class is focused on a single requirement. The class Lecturer in the previous figure exhibits low levels of cohesion as it has three attributes (roomNumber, roomLength and roomWidth and one operation calculate RoomSpace ()) that would be more appropriate in a class Room. The class Lecturer should only have attributes that describe a Lecturer object (e.g. lecturerName and lecturerAddress) and operations that use them.

Specialization Cohesion addresses the semantic cohesion of inheritance hierarchies. For example in the following figure all the attributes and operations of the Address base class are used by the derived classes - this hierarchy has high inheritance coupling. However, it is neither true that a person is a kind of address nor that a company is a kind of address. The example is only using inheritance as a syntactic structure for sharing attributes and operations. This structure has low specialization cohesion and is poor design. It does not reflect meaningful inheritance in the problem domain.



7. b) Explain about singleton structural and behavioral patterns. 6M

Singleton--------2M
Structural-------2M
Behavioural----2M

The singleton pattern is described below in more general language.

Name. Singleton.

Problem. How can a class be constructed that should have only one instance and that can be accessed globally within the application?

Context. In some applications it is important that a class have exactly one instance. A sales order processing application may be dealing with sales for one company. It is necessary to have a Company object that holds details of the company's name, address, taxation reference number and so on. Clearly there should only be one such object. Alternative forms of a singleton object may be required depending upon different initial circumstances.

Forces. One approach to making an object globally accessible is to make it a global variable but in general this is not a good design solution as it violates encapsulation. Another approach is not to create an object instance at all but to use class operations and attributes . However, this limits the extensibility of the model since polymorphic redefinition of class operations is not possible in all development environments .

Solution. Create a class with a class operation getInstance( ) , which, when the class is first accessed, creates the relevant object instance and returns the object identity to the client. On subsequent accesses of the getInstance() operation no additional instance is created but the object identity of the existing object is returned.

The Advantages and Disadvantages of singleton patterns.

+ It provides controlled access to the sole object instance as the Singleton class encapsulates the instance.

+ The namespace is not unnecessarily extended with global variables.

-Using the pattern introduces some additional message passing. To access the singleton instance the class scope method has to be accessed first rather than accessing the instance directly.

- The pattern limits the flexibility of the application. If requirements change and as a result the singleton class may have many instances then accommodating this new requirement necessitates significant modification to the system.

Structural patterns

Structural patterns address issues concerned with the way in which classes and objects are organized. Structural patterns offer effective ways of using object-oriented constructs such as inheritance, aggregation and composition to satisfy particular requirements. If there , a requirement for a particular aspect of the application to be extensible. In order to achieve

this, the application should be designed with constructs that minimize the sideeffects of future change. Alternatively it may be necessary to provide the same interface for a series of objects of different classes also.

Eg : Composite Pattern

Composite Pattern

To apply Composite structural pattern in a design for the Agate case study , consider if further work is required to design a multimedia application that can store and play components of an advert.

Here an advert is made up of sound clips and video clips each of which may be played individually or as part of an advert. The classes SoundClip and VideoClip have attributes and operations in common and it is appropriate that these classes are subclassed from MediaClip as shown below as MediaClip Inheritance hierachy.

Name. Composite.

Problem. There is a requirement to represent whole-part hierarchies so that both whole and part objects offer the same interface to client objects.

Context. In an application both composite and component objects exist that are required to offer the same behavior. Client objects should be able to treat composite or component objects in the same way. A commonly used example for the composite pattern is a graphical drawing package. Using this software package a user can create atomic objects like circle or square and can also group a series of atomic objects or composite objects together to make a new composite object. It should be possible to move or copy this composite object in exactly the same way as it is possible to move or copy an individual square or a circle.

Forces. The requirement that the objects, whether composite or component, offer the same interface suggests that they belong to the same inheritance hierarchy. This enables operations to be inherited and to be polymorphically redefined with the same signature. The need to represent whole-part hierarchies indicates the need for an aggregation structure.

Solution. The solution resolves the issues by combining inheritance and aggregation hierarchies. Both subclasses, Leaf and Composite, have a polymorphically redefined operation an Operation (). The Composite subclass also has additional operations to manage the aggregation hierarchy so that components may be added or removed.

Behavioural patterns addresses the problems that arise when assigning responsibilities to classes and when designing algorithms. Behavioural patterns specifies static relationships between objects and classes and how the objects of one class communicates with another. Behavioural patterns may use inheritance structures to spread behaviour across the subclasses or they may use aggregation and composition to build complex behaviour from simpler components.

Eg: The State pattern, uses both of these techniques.

State Pattern

Consider Agate case study to determine whether it has features that may satisfy the application of the state pattern or not. Identify whether there are any objects with significant state dependent behaviour or not. Among those Campaign objects will have behaviour that varies according to state , may be in any one of four main states, as shown below.

1.Commissioned State.

2.Active State.

3.Completed State

4.Paid State.

Clearly a Campaign object's state changes dynamically as the campaign progresses, thus necessitating changes in the behaviour of the object.

Name. State pattern.

Problem. An object exhibits different behaviour when its internal state changes making the object appear to change class at run-time.

Context. In some applications an object may have complex behaviour that is dependent upon its state. In other words the response to a particular message varies according to the object's state. One example is the calcCosts () operation in the Campaign class.

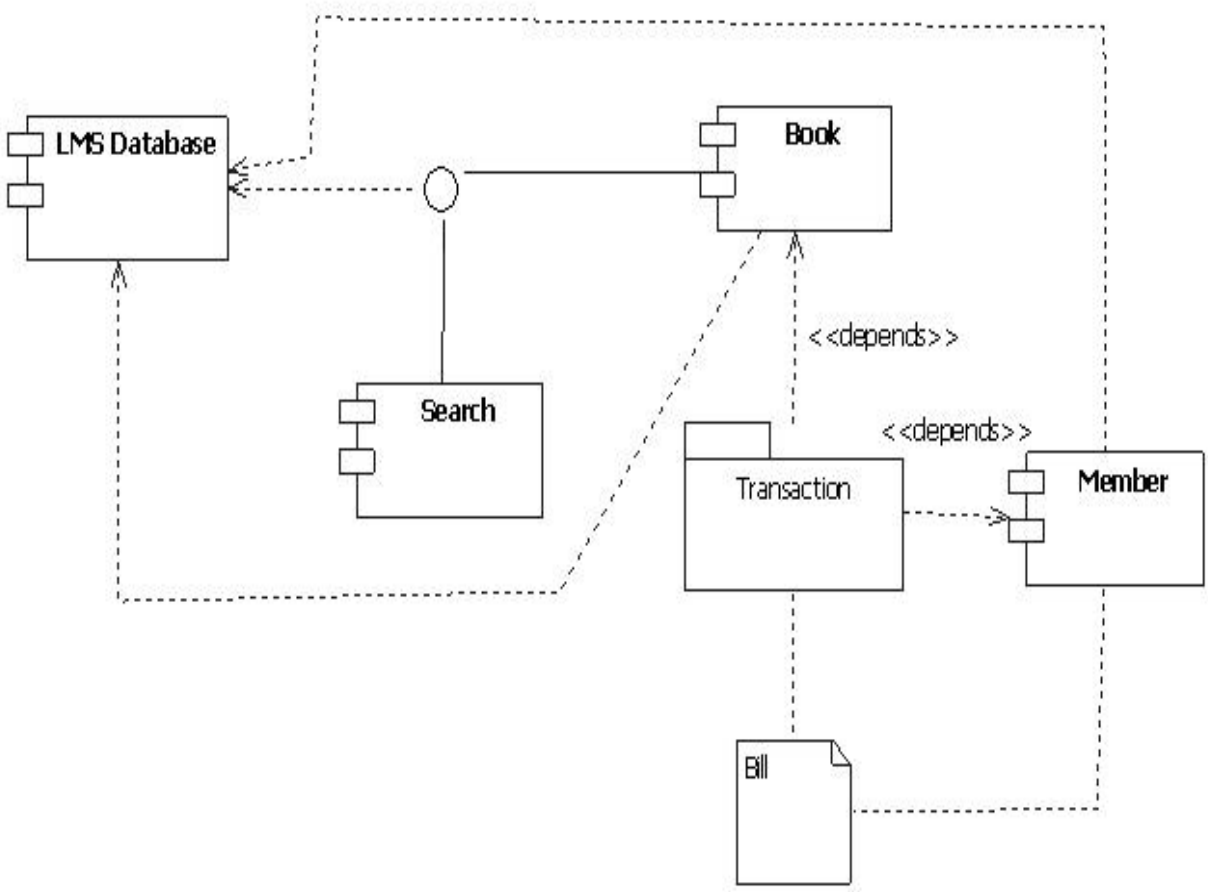
Forces. The object has complex behaviour that should be factored into less complex elements. One or more operations have behaviour that varies according to the state of the object. Typically the operation would have large, multi-part conditional statements depending on the state. One approach is to have separate public operations for each state but client objects would need to know the state of the object so that they could invoke the appropriate operation. of a separate private operation for each state may result in a large complex object that is difficult to construct, test and maintain.

UNIT-IV

8. a) D raw the component diagram for library management system? 6M

Any relevant example can be considered

Example:



8. b) Explain about DSDM and XP process models? 6M

DSDM MODEL----------4m
XP MODEL----------2m

The Dynamic Systems Development Method (DSDM) is a management and control framework for rapid application development (RAD). The distinction between RAD and prototyping is sometimes unclear. A RAD approach aims to build a working system rapidly while a prototyping approach also builds rapidly, but usually only produces a partially complete system, typically to confirm some aspect of the requirement. Because both approaches aim to build software quickly, similar development environments are used and one approach to prototyping continues the development of a prototype incrementally until it becomes a working system. In effect this is a RAD development approach.

DSDM is based upon the following nine principles.

1. Active user involvement is imperative. Many other approaches effectively restrict user involvement to requirements acquisition at the beginning of the project and acceptance testing at the end of the project. In DSDM users are members of the project team and include one known as an 'Ambassador' user.

2. DSDM teams are empowered to make decisions. A team can make decisions that refine the requirements and possibly even change them without the direct involvement of higher management.

3. The focus is on frequent product delivery. A team is geared to delivering products in an agreed time period and it selects the most appropriate approach to achieve this.

4. The essential criterion for acceptance of a deliverable is fitness for business purpose. DSDM is geared to delivering the essential functionality at the specified time.

5. Iterative and incremental development is necessary to converge on an accurate business solution. Incremental development allows user feedback to inform the development of later increments. The delivery of partial solutions is considered acceptable if they satisfy an immediate and urgent user need. These solutions can be refined and further developed later.

6. All changes during development are reversible. If the iterative development follows an inappropriate development path then it is necessary to return to the last point in the development cycle that was considered appropriate. Changes are limited within a particular increment.

7. Requirements are initially agreed at a high level. Once requirements are fixed at a high level they provide the objectives for prototyping. The requirements can then be investigated in detail by the DSD M teams to determine the best way to achieve them. Normally the scope of the high level requirements is not changed significantly.

8. Testing is integrated throughout the life cycle. Since a partially complete system may be delivered it must be tested during development, rather than after completion. Each software component is tested by the developers for technical compliance and by user team members for functional appropriateness.

9. A collaborative and co-operative approach between all stakeholders is essential. The emphasis here is on the inclusion of all stakeholders in a collaborative development process. Stakeholders not only include team members, but others such as resource managers and the quality assurance team.

Extreme Programming:

Extreme Programming (XP) is a novel combination of elements of best practice in systems development. It incorporates a highly iterative approach to development. It has become well known in a relatively short period of time for its use of ***pair programming***

though it encompasses various other important ideas. Pair programming involves writing the program code in pairs and not individually.

Beck identified the following four underlying principles of XP as communication, simplicity, feedback and courage.

Communication. Poor communication is a significant factor in failing projects, XP highlights the importance of good communication among developers and between developers and users.

Simplicity. Software developers are sometimes tempted to use technology for technology's sake rather than seeking the simplest effective solution. Developers justify complex solutions as a way of meeting possible future requirements. XP focuses on the simplest solution for the immediate known requirements.

Feedback. Unjustified optimism is common in systems development. Developers tend to underestimate the time required to complete any particular programming task. This results in poor estimates of project completion, constant chasing of unrealistic deadlines, stressed developers and poor product quality. Feedback in XP is geared to giving the developers frequent and timely feedback from users and also in terms of test results. Work estimates are based on the work actually completed in the previous iteration.

Courage. The exhortation to be courageous urges the developer to throwaway code that is not quite correct and start again rather than trying to fix the unfixable. Essentially the developer has to leave unproductive lines of development despite personal investment in the ideas.

9. a) Explain about prototyping the user interface. 6M

Users of an information system need to interact with it in some way.

User interface (UI) prototyping is an iterative development technique in which users are actively involved in the mocking-up of the UI for a system. UI prototypes have several purposes:

i. As an analysis artifact that enables you to explore the problem space with your stakeholders.
ii. As a design artifact that enables you to explore the solution space of your system.
iii. A basis from which to explore the usability of your system.
iv. A vehicle for you to communicate the possible UI design(s) of your system.
v. A potential foundation from which to continue developing the system (if you intend to throw the prototype away and start over from scratch then you don’t need to invest the time writing quality code for your prototype).

- **Work with the real users**. The best people to get involved in prototyping are the ones who will actually use the application when it is done. These are the people who have the most to gain from a successful implementation; these are the people who know their own needs best. Follow the Agile Modeling (AM) practice Active Stakeholder Participation.
- **Get your stakeholders to work with the prototype**. Just as if you want to take a car for a test drive before you buy it, your users should be able to take an application for a test drive before it is developed. Furthermore, by working with the prototype hands-on, they can quickly determine whether the system will meet their needs. A good

approach is to ask them to work through some use case scenarios using the prototype as if it were the real system.

- **Understand the underlying business**. You need to understand the underlying business before you can develop a prototype that will support it. In other words, you need to base your UI prototype on your requirements. The more you know about the business, the more likely it is you can build a prototype that supports it. Once again, active stakeholder participation is critical to your success.
- **You should only prototype features that you can actually build**. Christmas wish lists are for kids. If you cannot possibly deliver the functionality, do not prototype it.
- **You cannot make everything simple**. Sometimes your software will be difficult to use because the problem it addresses is inherently difficult. Your goal is to make your user interface as easy as possible to use, not simplistic.
- **It's about what you need**. Constantine and Lockwood differentiate between the concepts of WYSIWYG, "What You See Is What You Get," and WYSIWYN, ""What You See Is What You Need." Their point is a good user interface fulfills the needs of the people who work with it. It isn't loaded with a lot of interesting, but unnecessary, features.

9. b) What is reuse? Explain the strategy planned for reuse. 6M

Reuse definition------1M
Strategy for reuse----5M

Reuse: Reusability is the use of existing assets in some form within the software product development process.

Reusable software components are designed to apply the power and benefit of reusable, interchangeable parts from other industries to the field of software construction. Other industries have long profited from reusable components

Planning a strategy for Reuse

The following are two approaches to the introduction of a reuse strategy.

The SELECT Perspective

Allen and Frost describes the SELECT Perspective approach to the developers of reusable components. At the level of practical techniques, this includes guidelines for the modelling of business-oriented components and for wrapping legacy software in component wrappers. They distinguish between reuse at the level of component packages, which consist of executable components grouped together, and service packages, which are abstractions of components that group together business services.

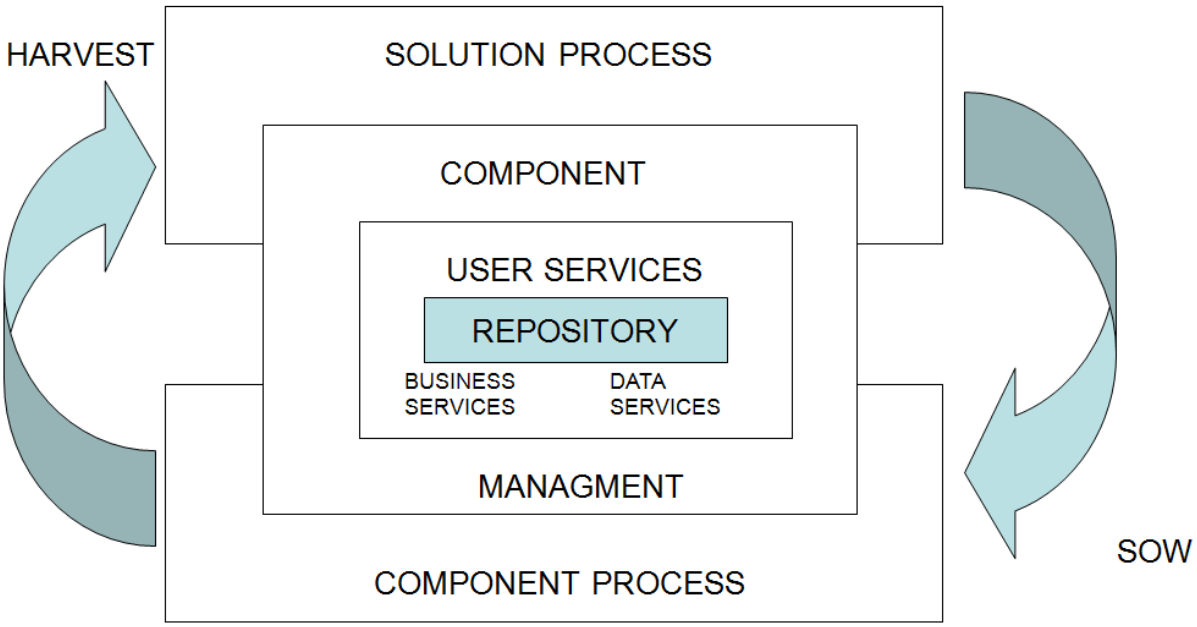
The focus of this approach is to identify the services that belong together and classes that implement them. Service classes in a single package should have a high level of internal interdependency and minimal coupling to classes in other packages .

In order to develop reusable components while achieving the development of a system to meet users' needs, the Perspective approach breaks the development process into two parts:

1. The solution process

2. The component process.

These two parts run in parallel and feed off each other.



The SELECT Perspective service-based process

- The solution process focuses on specific business needs and delivering services to meet the users' requirements. Its products have immediately definable business value. During the solution process, developers will draw on the component process in their search for reusable components that can be applied to the project.

- The component process focuses on developing reusable components in packages that group together families of classes to deliver generic business services. During the component process, the developers produce components that can be reused in the solution process. The component process also searches out opportunities to reuse services from existing legacy systems and legacy databases and from other packages of components.

  Software support is needed for effective component reuse to take place. This support takes the form of repository-based component management software. Components are placed in the repository as a means of publishing them and making them available to other users. The repository is made up of catalogues and the catalogues contain details of components, their specifications and their interfaces. Component management software tools provide the functionality for adding components to the repository and for browsing and searching for components. Component management software may be integrated with CASE tools to allow the storage of analysis and design models as well as source code and executables.