


CMR INSTITUTE OF TECHNOLOGY		USN								
Internal Assessment Test - III										
Sub:	Object Oriented Modeling and Design						Code:	16MCA51		
Date:	19.11.2018	Duration:	90 mins	Max Marks:	50	Sem:	V	Branch:	MCA	
Answer Any One FULL Question from each part.										
								Mark s	OBE CO RBT	
<b>Part - I</b>										
1	(a)	What is software control strategy? Explain the different types of software control strategy.					[6]	CO4	L1,L4	
	(b)	Draw the diagram for architecture of ATM system					[4]	CO3	L1	
2	(a)	Show how the system is broken into subsystems. Explain the various layers of the sub system.					[5]	CO4	L3	
	(b)	Explain about the making of a reuse plan.					[5]	CO4	L4	
<b>Part – II</b>										
3	(a)	Describe in detail about the design optimization.					[8]	CO5	L2	
	(b)	Summarize on reification of behavior.					[2]	CO5	L2	
4	(a)						[8]	CO5	L2	
	(b)						[2]	CO5	L2	
<b>Part - III</b>										
5	(a)	Describe the various steps to construct an application interaction model					[10]	CO3	L1	
6	(a)	Explain the common architectural styles					[10]	CO4	L4	
<b>Part – IV</b>										
7	(a)	Discuss the steps to construct a domain class model, with an example of ATM					[10]	CO3	L2	
8	(a)	Draw the activity diagram for card verification					[4]	CO3	L1	
	(b)	Briefly explain the application state model					[6]	CO3	L2	
<b>Part – V</b>										
9	(a)	Elaborate and explain the questions to be answered for a good system concept.					[4]	CO3	L2	
	(b)	Write in detail about the software development stages.					[6]	CO3	L2	
10(a)		Explain the states and events. Discuss the different kinds of events					[5]	CO2	L3	
	(b)	What do you mean by Swim Lane? Draw an activity diagram with Swim Lanes for servicing an airplane.					[5]	CO2	L1	

**1a. What is software control strategy? Explain the different types of software control strategy. [6]**  
 Choosing a Software Control Strategy

- To choose a single control style for the whole system.
- Two kinds of control flows:

**1. External Control**

- Concerns the flow of externally visible events among the objects in the system.
- Three kinds:

**a. Procedure-driven Control**

- Control resides within the program code
- Procedure request external input and then wait for it
- When input arrives, control resumes with in the procedure that made the call.
- Advantage:
  - Easy to implement with conventional languages
  - Disadvantage:
    - The concurrency inherent in objects are to mapped into a sequential flow of control.
    - Suitable only if the state model shows a regular alternation of input and output events.
      - C++ and Java are procedural languages.

They fail to support the concurrency inherent in objects.

**b. Event-driven Control**

- Control resides within a dispatcher or monitor that the language, subsystem, or operating system provides.
- The dispatcher calls the procedures when the corresponding events occur. Event-driven systems permit more flexible control than procedure-driven systems.

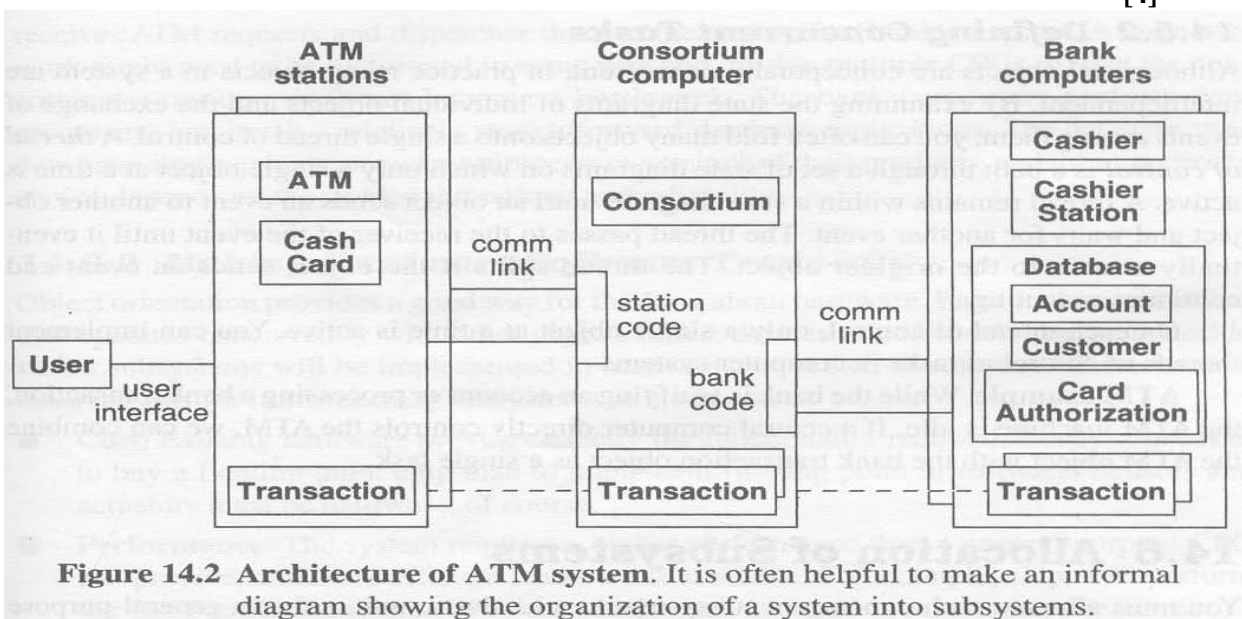
**c. Concurrent Control**

Control resides concurrently in several independent objects, each a separate task. Such a system implements events directly as one way message between object.

**2. Internal Control**

- Refer to the flow of control within a process.
- To decompose a process into several tasks for logical clarity or for performance.
- Three kinds:
  - Procedure calls : under the direction of the program and can be structured for convenience.
  - Quasi concurrent inter task call: Multiple address spaces or call stacks exist but only a single thread of control can be active at once.
  - Current inter task calls

**1b. Draw the diagram for architecture of ATM system [4]**



**2a. Show how the system is broken into subsystems. Explain the various layers of the sub system. [5]**

Divide the system into pieces . Each piece of a system is called subsystem. A subsystem is a group of classes, associations, operations, events, and constrains. A subsystem is usually identified by the services it provides. Each subsystem has a well-defined interface to the rest of the system.

The relation between two subsystems can be – Client-server relationship – Peer-to-peer relationship

The decomposition of systems into subsystems is organized as a sequence of

- Horizontal layers,
- Vertical partitions, or
- Combination of layers and partitions.

**1. LAYERS :** Each built in terms of the ones below it. The objects in each layer can be independent.

E.g. A client-server relationship. Problem statement specifies only the top and bottom layers:

- The top is the desired system.
- The bottom is the available resources.

The intermediate layers is than introduced. • Two forms of layered architectures:

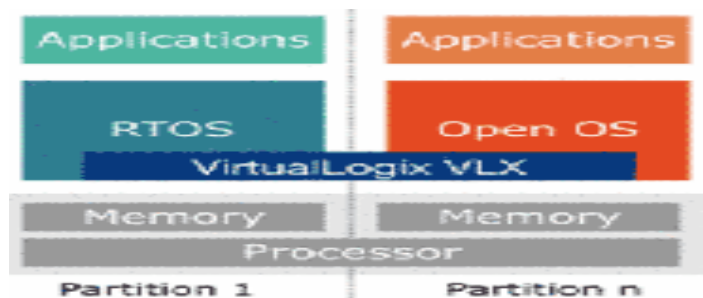
- Closed architecture • Each layer is built only in terms of the immediate lower layer.
- Open architecture • A layer can use features on any lower layer to any depth
- Do not observe the principle of information hiding.

**2. PARTITIONS**

Vertically divided into several subsystems .Independent or weakly coupled Each providing one kind of service. E.g. A computer operating system includes

- File system
- Process control
- Virtual memory management

**3. COMBINATION OF LAYERS AND PARTITIONS**



**2b. Explain about the making of a reuse plan. [5]**

Two aspects of reuse:

- Using existing things
- Creating reusable new things

Reusable things include: Models, Libraries, Frameworks ,Patterns

**1. Libraries** A library is a collection of classes that are useful in many contexts.

Qualities of “Good” class libraries:

- *Coherence* – well focused themes
- *Completeness* – provide complete behaviour
- *Consistency* - polymorphic operations should have consistent names and signatures across classes.
- *Efficiency* – provide alternative implementations of algorithms.
- *Extensibility* – define subclasses for library classes
- *Genericity* – parameterized class definitions

Problems limit the reuse ability:

- Argument validation • Validate arguments by collection or by individual
- Error Handling • Error codes or errors
- Control paradigms • Event-driven or procedure-driven control
- Group operations
- Garbage collection
- Name collisions

**2. Frameworks** A framework is a skeletal structure of a program that must be elaborated to build a complete application. Frameworks class libraries are typically application specific and not suitable for general use.

**3. Patterns** A pattern is a proven solution to a general problem. There are patterns for analysis, architecture, design, and implementation.

Benefits of Pattern

1. A pattern is considered by others and has already been applied to past problems .
2. When you use patterns, you tap into a language that is familiar to many developers
3. Patterns are prototypical model fragments that distil some of the knowledge of experts.

**Pattern vs. Framework**

1. A pattern is typically a small number of classes and relationships.
2. A framework is much broader in scope and covers an entire subsystem or application.

**3a. Describe in detail about the design optimization.**

[8]

The design model builds on the analysis model. The analysis model captures the logic of a system, while the design model adds development details. You can optimize the inefficient but semantically correct analysis model to improve performance, but an optimized system is more obscure and less likely to be reusable. You must strike an appropriate balance between efficiency and clarity. Design optimization involves the following tasks.

- Provide efficient access paths.
- Rearrange the computation for greater efficiency.
- Save intermediate results to avoid recomputation.

**15.7.1 Adding Redundant Associations for Efficient Access**

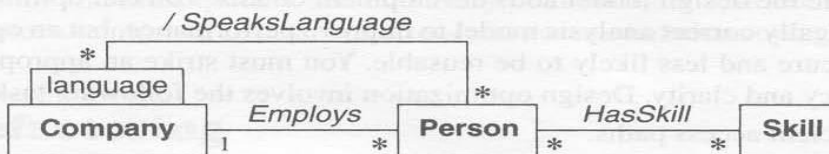
Redundant associations are undesirable during analysis because they do not add information. Design, however, has different motivations and focuses on the viability of a model for implementation.

For an example, consider the design of a company’s employee skills database. Figure 15.5 shows a portion of the analysis class model. The operation *Company.findSkill()* returns a set of persons in the company with a given skill. For example, an application might need all the employees who speak Japanese.



**Figure 15.5** Analysis model for person skills. Derived data is undesirable during analysis because it does not add information.

The derived association does not add any information but permits fast access to employees who speak a particular language. Indexes incur a cost: They require additional memory and must be updated whenever the base associations are updated. As the designer, you decide when it is worthwhile to build indexes. Note that if most queries return a high fraction of the objects in the search path, then an index really does not save much.



**Figure 15.6** Design model for person skills. Derived data is acceptable during design for operations that are significant performance bottlenecks.

Start by examining each operation and see what associations it must traverse to obtain its information. Next, for each operation, note the following.

- **Frequency of access.** How often is the operation called?
- **Fan-out.** What is the “fan-out” along a path through the model? Estimate the average count of each “many” association encountered along the path. Multiply the individual fan-outs to obtain the fan-out of the entire path, which represents the number of accesses on the last class in the path. “One” links do not increase the fan-out, although they increase the cost of each operation slightly; don’t worry about such small effects.
- **Selectivity.** What is the fraction of “hits” on the final class—that is, objects that meet selection criteria and are operated on? If the traversal rejects most objects, then a simple nested loop may be inefficient at finding target objects.

## 15.7.2 Rearranging Execution Order for Efficiency

After adjusting the structure of the class model to optimize frequent traversals, the next thing to optimize is the algorithm itself. One key to algorithm optimization is to eliminate dead paths as early as possible. For example, suppose an application must find all employees who speak both Japanese and French. Suppose 5 employees speak Japanese and 100 speak French; it is better to test and find the Japanese speakers first, then test if they speak French. In general, it pays to narrow the search as soon as possible. Sometimes you must invert the execution order of a loop from the original specification.

### 15.7.3 Saving Derived Values to Avoid Recomputation

Sometimes it is helpful to define new classes to cache derived attributes and avoid recomputation. You must update the cache if any of the objects on which it depends are changed. There are three ways to handle updates.

- **Explicit update.** The designer inserts code into the update operation of source attributes to explicitly update the derived attributes that depend on it.
- **Periodic recomputation.** Applications often update values in bunches. You could recompute all the derived attributes periodically, instead of after each source change. Periodic recomputation is simpler than explicit update and less prone to bugs. On the other hand, if the data changes incrementally a few objects at a time, full recomputation can be inefficient.
- **Active values.** An *active value* is a value that is automatically kept consistent with its source values. A special registration mechanism records the dependency of derived attributes on source attributes. The mechanism monitors the values of source attributes and updates the values of the derived attributes whenever there is a change. Some programming languages provide active values.

### 3b. Summarize on reification of behavior.

[2]

Behaviour written in code is rigid; you can execute but cannot manipulate it at run time. If you need to store, pass, or modify the behaviour at run time, you should reify it.

### 4a. Discuss the various steps to design algorithms

[8]

Formulate an algorithm for each operation. The analysis specification tells what the operation does for its Clients. The algorithm show how it is done

#### 1. Choosing algorithms (Choose algorithms that minimize the cost of implementing operations)

When efficiency is not an issue, you should use simple algorithms. Typically, 20% of the operations consume 80% of execution time. Considerations for choosing alternative algorithms

- a. Computational complexity
- b. Ease of implementation and understandability
- c. Flexibility

Simple but inefficient. Complex efficient

#### 2. Choosing Data Structures (select data structures appropriate to the algorithm)

- a. Algorithms require data structures on which to work.
- b. They organize information in a form convenient for algorithms.
- c. Many of these data structures are instances of container classes.
- d. Such as arrays, lists, queues, stacks, set...etc.

#### 3. Defining New Internal Classes and Operations

- a. To invent new, low-level operations during the decomposition of high-level operations.
- b. The expansion of algorithms may lead you to create new classes of objects to hold intermediate results.

ATM Example:

- i. Process transaction uses case involves a customer receipt.
- ii. A Receipt class is added.

#### 4. Assigning Operations to Classes (assign operations to appropriate classes)

- a. How do you decide what class owns an operation?
  - i. Receiver of action: To associate the operation with the target of operation, rather than the initiator.
  - ii Query vs. update : The object that is changed is the target of the operation
  - iii Focal class : Class centrally located in a star is the operation's target
  - iv. Analogy to real world

### 4b. Summarize on recursive downward.

[2]

To organize operations as layers. Operations in higher layers invoke operations in lower layers.

Two ways of downward recursion:

**Functionality Layers :** Take the required high-level functionality and break it into lesser operations. Make sure you combine similar operations and attach the operations to classes. An operation should be coherent meaningful, and not an arbitrary portion of code.

ATM eg., use case decomposed into responsibilities .Resulting operations are assigned to classes. If it is not satisfied rework them.

**Mechanism Layers :** Build the system out of layers of needed support mechanisms. These mechanisms don't show up explicitly in the high-level responsibilities of a system, but they are needed to make it all work.

E.g. Computing architecture includes Data structures, algorithms, and control patterns.

A piece of software is built in terms of other, more mechanisms than itself.

## 5a. Describe the various steps to construct an application interaction model.

[10]

### Application Interaction Model - steps to construct model

1. Determine the system boundary
2. Find actors
3. Find use cases
4. Find initial and final events
5. Prepare normal scenarios
6. Add variation and exception scenarios
7. Find external events
8. Prepare activity diagrams for complex use cases.
9. Organize actors and use cases
10. Check against the domain class model

#### 1. Determine the system boundary

Determine what the system includes. What should be omitted? Treat the system as a black box.

ATM example:

**2.Find actors** The external objects that interact directly with the system. They are not under control of the application. Not individuals but archetypical behaviour.

ATM Example: Customer, Bank, Consortium

**3.Find use cases** For each actor, list the different ways in which the actor uses the system. Try to keep all of the uses cases at a similar level of detail.

- apply for loan
- withdraw the cash from savings account
- make withdrawal

Use Case for the ATM

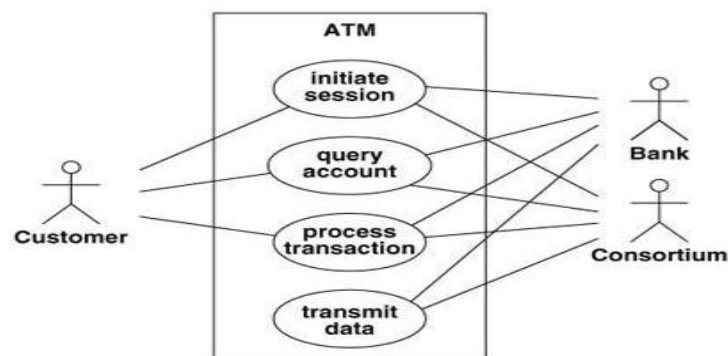


Figure 13.1 Use case diagram for the ATM. Use cases partition the

**4.Find initial and final events :**Finding the initial and final events for each use case .To understand the behaviour clearly of system .Execution sequences that cover each use case

Initial events may be

- a. A request for the service that the use case provides
- b. An occurrence that triggers a chain of activity

#### 5. Prepare normal scenarios

For each use case, prepare one or more typical dialogs. A scenario is a sequence of events among a set of interacting objects. Sometimes the problem statement describes the full interaction sequence

#### 6. Normal ATM scenarios

**7. Add variation and exception scenarios**

Special cases :Omitted input E.g., maximum values, minimum value

Error cases: E.g. Invalid values, failures to respond

Other cases E.g. Help requests, status queries

**8. Find external events:** The external events include All inputs, decisions, interrupts, and Interactions to or from users or external devices.

- An event can trigger effects for a target object.
- Use scenarios for normal events

Sequence diagram

- Prepare a sequence diagram for each scenario.
- The sequence diagram captures the dialog and interplay between actors. The sequence diagram clearly shows the sender and receiver of each event ATM Example

**9. Activity Diagram** Activity diagram shows behaviours like alternatives and decisions.

- Prepare activity diagrams for complex use cases.
- Appropriate to document business logic during analysis
- Do not use activity diagram as an excuse to begin implementation.

**10. Organize actors and use cases** Organize use cases with relationships – Include, extend, and generalization

- Organize actors with generalization.

**11. Checking Against the Domain Class Model**

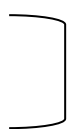
- The application and domain models should be mostly consistent.
- The actors, use cases, and scenarios are all based on classes and concepts from the domain model.
- Examine the scenarios and make sure that the domain model has all the necessary data.
- Make sure that the domain model covers all event parameters.

**6a. Explain the common architectural styles.**

[10]

Several prototypical architectural styles are common in existing system.

- Some kinds of systems:
  - Batch transformation -----> Functional transformations
  - Continuous transformation
  - Interactive interface
  - Dynamic simulation
  - Real-time system
  - Transaction manager -----> Database system



Time-dependent systems

**Batch transformation**

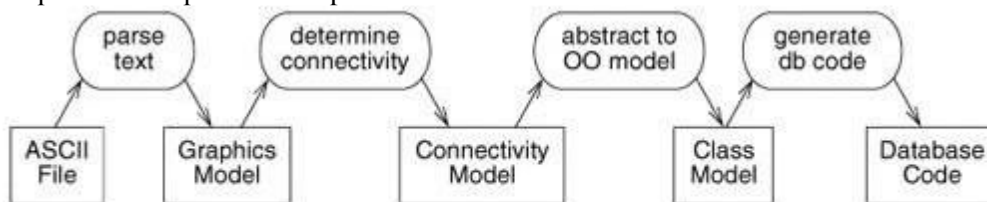
- Perform sequential computation.
- The application receives the inputs, and the goal is to compute an answer.
- Does not interact with the outside world

• E.g.

- Compiler
- Payroll processing
- VLSI automatic layout
- :

• The most important aspect is to define a clean series of steps

• Sequence of steps for a compiler



**Figure 14.5 Sequence of steps for a compiler.** A batch transformation is a sequential input-to-output transformation that does not interact with the outside world.

The steps in designing a batch transformation are as follows

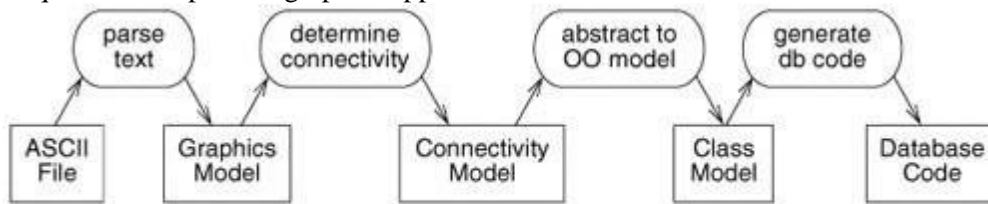
- Break the overall transformation into stages, with each stage performing one part of the transformation.
- Prepare class models for the input, output and between each pair of successive stages. Each stage knows only about the models on either side of it.
- Expand each stage in turn until the operations are straightforward to implement.
- Restructure the final pipeline for optimization.

Continuous transformation

- The outputs actively depend on changing inputs.
- Continuously updates the outputs (in practice discretely )



- E.g.
- Signal processing
- Windowing systems
- Incremental compilers
- Process monitoring system
- Sequence of steps for a graphics application



**Figure 14.5 Sequence of steps for a compiler.** A batch transformation is a sequential input-to-output transformation that does not interact with the outside world.

- Steps in designing a pipeline for a continuous transformation are as follows
  - o Break the overall transformation into stages, with each stage performing one part of the transformation.
  - o Define input, output and intermediate models between each pair of successive stages as for the batch transformation
  - o Differentiate each operation to obtain incremental changes to each stage.
  - o Add additional intermediate objects for optimization.

### Interactive interface

- Dominated by interactions between the system and external agents.

Steps in designing an interactive interface are as follows

- Isolate interface classes from the application classes
- Use predefined classes to interact with external agents
- Use the state model as the structure of program
- Isolate physical events from logical events.
- Fully specify the application functions that are invoked by the interface

### Dynamic simulation

- Models or tracks real-world objects.
- Steps in designing a dynamic simulation
  - Identify active real-world objects from the class model.
  - Identify discrete events
  - Identify continuous dependencies
  - Generally simulation is driven by a timing loop at a fine time scale

### Real-time system

- An interactive system with tight time constraints on actions.

### Transaction manager

- Main function is to store and retrieve data.
- Steps in designing an information system are as follows
  - Map the class model to database structures.
  - Determine the units of concurrency
  - Determine the unit of transaction
  - Design concurrency control for transactions

### 7a. Discuss the steps to construct a domain class model, with an example of ATM.

[10]

Find classes

Prepare a data dictionary

Find associations

Find attributes of objects and links

Organize and simplify classes using inheritance

Verify that access paths exist for likely queries

Iterate and refine the model

Reconsider the level of abstraction

Group classes into packages.



Finding Classes:- Classes often correspond to nouns for example . In the statement “a reservation system to sell tickets to performances at various theaters tentative classes would be Reservation , System , Ticket ,Performances and Theater

Keeping the right classes:

We need to discard the unnecessary and incorrect classes:

Redundant classes: if two classes express same

Irrelevant classes: It has little or nothing to do with the problem

Vague class: too broad in scope

Attributes: describe individual objects

Operations

Roles

Implementation constructs: CPU, subroutine, process and algorithm

Derived classes: omit class that can be derived from other class

Prepare data dictionary: Information regarding the data is maintained

Finding Associations

Keeping the right associations:

- 1.Associations between eliminated classes
- 2.Irrelevant or implementation associations
- 3.Actions
- 4.Ternary association
- 5.Derived Association
- 6.Misnamed associations
- 7.Association end names
- 8.Qualified associations
- 9.Multiplicity
- 10.Missing Associations
- 11.Aggregation

Finding attributes:

Keeping the right attributes:

Refining with inheritance :

1. Bottom-up generalization
2. Top-down generalization
3. Generalization vs enumeration
4. Multiple inheritance
5. Similar associations.
6. Adjusting the inheritance level

Testing Access paths

Iterating a Class Model : Several signs of missing classes

- **Asymmetries in associations and generalizations.** Add new classes by analogy.
- **Disparate attributes and operations on a class.** Split a class so that each part is coherent.
- **Difficulty in generalizing cleanly.** One class may be playing two roles. Split it up and one part may then fit in cleanly.
- **Duplicate associations with the same name and purpose.** Generalize to create the missing superclass that unites them.
- **A role that substantially shapes the semantics of a class.** Maybe it should be a separate class. This often means converting an association into a class. For example, a person

Shifting the level of abstraction

Grouping Classes into packages.

8a. Draw the activity diagram for card verification

[4]

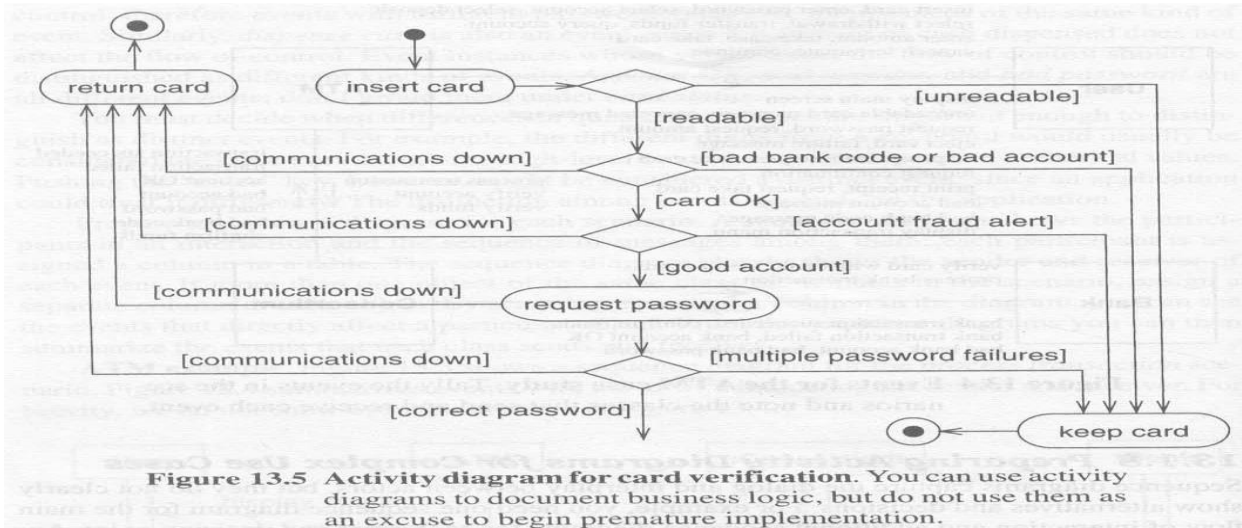


Figure 13.5 Activity diagram for card verification. You can use activity diagrams to document business logic, but do not use them as an excuse to begin premature implementation.

8b. Briefly explain the application state model

[6]

Application State Model

- The application state model focuses on application classes
- Augments the domain state model

Application State Model- steps

1. Determine Application Classes with States
2. Find events
3. Build state diagrams
4. Check against other state diagrams
5. Check against the class model
6. Check against the interaction model

1. Determine Application Classes with States

- Good candidates for state models
  - User interface classes
  - Controller classes • ATM example
  - The controllers have states that will elaborate.

2. Find events

- Study scenarios and extract events.
- In domain model
  - Find states and then find events
- In application model
  - Find events first, and then find states • ATM example
  - Revisit the scenarios, some events are:
    - *Insert card, enter password, end session and take card.*

3. Building State Diagrams

- To build a state diagram for each application class with temporal behaviour.
- Initial state diagram
  - Choose one of these classes and consider a sequence diagram.
  - The initial state diagram will be a sequence of events and states.
  - Every scenario or sequence diagram corresponds to a path through the state diagram.
- Find loops
  - If a sequence of events can be repeated indefinitely, then they form a loop.
- Merge other sequence diagrams into the state diagram.
- After normal events have been considered, add variation and exception cases.
- The state diagram of a class is finished when the diagram covers all scenarios and the diagram handles all events that can affect a state.
- Identify the classes with multiple states
- Study the interaction scenarios to find events for these classes

- Reconcile the various scenarios
  - Detect overlap and closure of loops
4. check against other state diagrams
- Every event should have a sender and a receiver.
  - Follow the effects of an input event from object to object through the system to make sure that they match the scenarios.
  - Objects are inherently concurrent.
  - Make sure that corresponding events on different state diagrams are consistent. • ATM example
  - The *SessionController* initiates the *TransactionController*,
  - The termination of the *TransactionController* causes the *SessionController* to resume.
5. Check against the class model
- ATM example
  - Multiple ATMs can potentially concurrently access an account.
  - Account access needs to be controlled to ensure that only one update at a time is applied.
6. Check against the interaction model
- Check the state model against the scenarios of the interaction model.
  - Simulate each behavior sequence by hand and verify the state diagrams.
  - Take the state model and trace out legitimate paths.
- Adding Operations
- Operations from the class model
  - Operations from use cases
  - Shopping-list operations
  - Simplifying operations
- Operations from the class model
- The reading and writing of attribute values and association links.
  - Need not show them explicitly.
- Operations from use cases
- Use cases lead to activities.
- Many of these activities correspond to operations on the class model. • ATM example
  - *Consortium* □ *verifyBankCode*.
  - *Bank* □ *verifyPassword*.
  - *ATM* □ *verifyCashCard*

**9a. Elaborate and explain the questions to be answered for a good system concept.**

[4]

A good system concept must answer the following questions.

- Who is the application for ?
- Understand the stakeholders of the system;  
Usually two important ones are:  
Financial sponsor (client)  
End Users
- What problems will it Solve?
- Bound the size of effort and scope of system  
Determine what feature is in and what is out
- Where will it be used ?
- Characterize the environment the system will be used, e,g  
Mission-critical?  
Experimental?  
Enhancements to existing system?  
For commercial products, characterize the customer base.
- When is it needed?
- Feasible time ( $T_f$ )  
The time in which the system can be developed within the constraints of cost and available resource  
Required time ( $T_r$ )  
The time that the system is needed to meet the business goals  
If ( $T_r < T_f$ ), work with technologists and business experts to trim the system
- Why is it needed ?
- Prepare a business case  
Financial justification including cost, tangible and intangible benefits, risks and alternatives.

For a commercial product, estimate the number of units you can sell and determine a reasonable price.

- How will it work

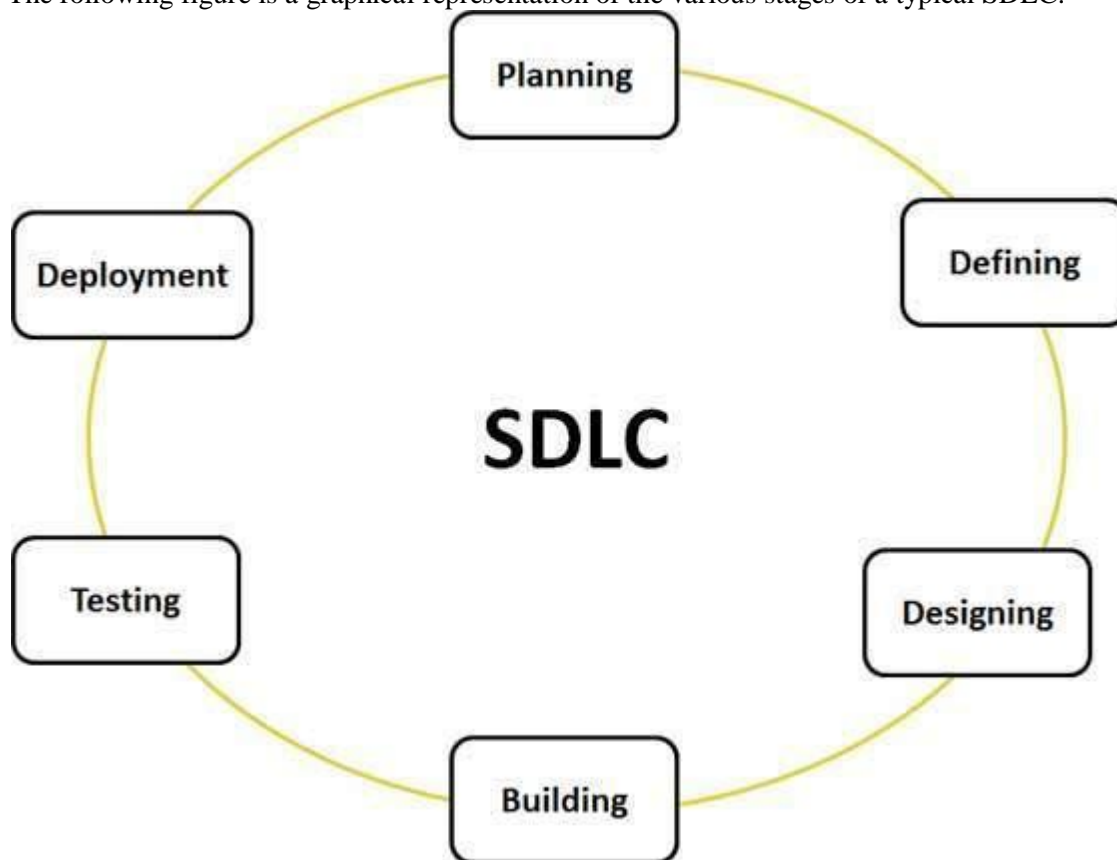
Investigate feasibility of the problem

Build prototype, if it helps clarifying a concept or removing a technological risk

### 9b. Write in detail about the software development stages.

[6]

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development life cycle consists of the following stages:

#### Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational, and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

#### Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through .SRS. . Software Requirement Specification document which consists of all the product requirements to be designed and developed during the project life cycle.

#### Stage 3: Designing the product architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity , budget and time constraints , the best design approach is selected for the product. A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minute test of the details in DDS.

#### Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers have to follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers etc are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java, and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

**Stage 5: Testing the Product**

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However this stage refers to the testing only stage of the product where products defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

**Stage 6: Deployment in the Market and Maintenance**

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometime product deployment happens in stages as per the organizations. business strategy. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

**10a.Explain the states and events. Discuss the different kinds of events [5]**

**States:** A state is an abstraction of the values and links of an object. Sets of values and links are grouped together into a state according to the gross behavior of objects

UML notation for state- a rounded box Containing an optional state name, list the state name in boldface, center the name near the top of the box, capitalize the fist letter.

- Ignore attributes that do not affect the behavior of the object.
- The objects in a class have a finite number of possible states.
- Each object can be in one state at a time.
- A state specifies the response of an object to input events.
- All events are ignored in a state, except those for which behavior is explicitly prescribed.

**Event vs. States :**

Event represents points in time.

State represents intervals of time.

A state corresponds to the interval between two events received by an object.

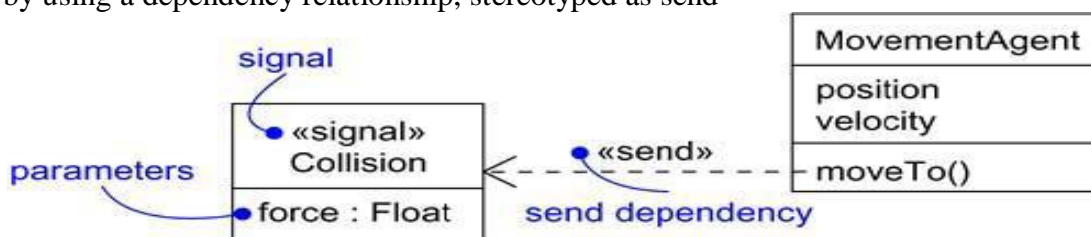
The state of an object depends on past events.

Both events and states depend on the level of abstraction.

An **event** is an occurrence at a point in time, such as user depresses left button of mouse. An event happens instantaneously with regard to the time scale of an application. Events cause state changes which is shown in State Diagrams

**Signal Event**

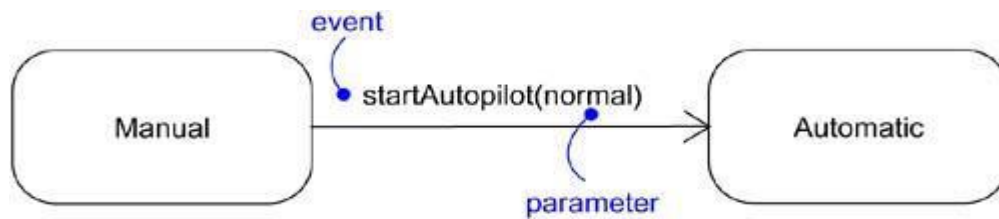
- a signal event represents a named object that is dispatched (thrown) asynchronously by one object and then received (caught) by another. Exceptions are an example of internal signal
- a signal event is an asynchronous event
- signal events may have instances, generalization relationships, attributes and operations. Attributes of a signal serve as its parameters
- A signal event may be sent as the action of a state transition in a state machine or the sending of a message in an interaction
- signals are modeled as stereotyped classes and the relationship between an operation and the events by using a dependency relationship, stereotyped as send





## Call Event

- a call event represents the dispatch of an operation
- a call event is a synchronous event



## Time and Change Events

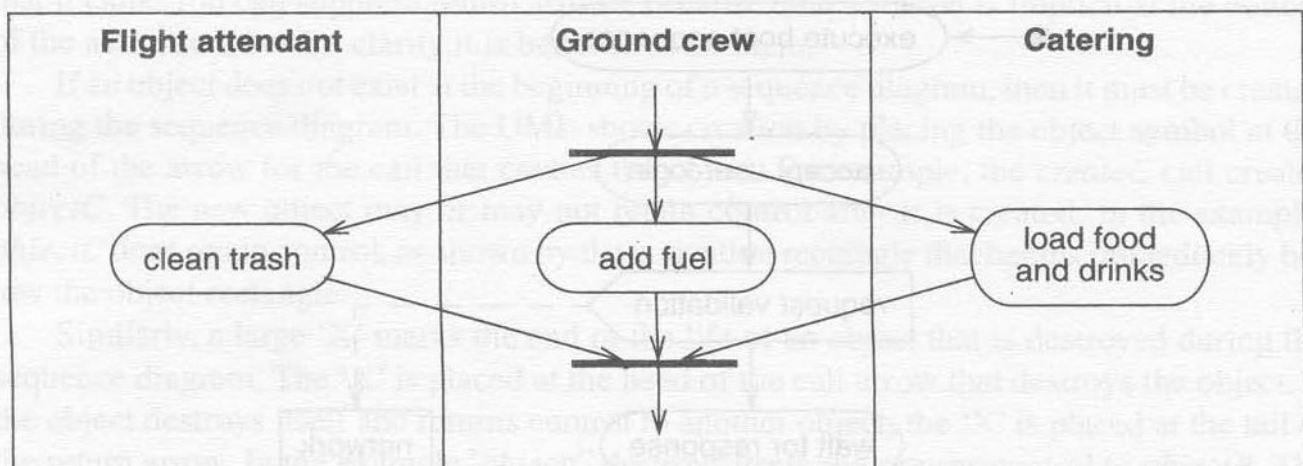
- A *time event* is an event that represents the passage of time.
- modeled by using the keyword 'after' followed by some expression that evaluates to a period of time which can be simple or complex.
- A *change event* is an event that represents a change in state or the satisfaction of some condition
- modeled by using the keyword 'when' followed by some Boolean expression

**10b. What do you mean by Swim Lane? Draw an activity diagram with Swim Lanes for servicing an airplane. [5]**

When

the design of the system is complete, the activity will be assigned to a person, but at a high level it is sufficient to partition the activities among organizations.

You can show such a partitioning with an activity diagram by dividing it into columns and lines. Each column is called a *swimlane* by analogy to a swimming pool. Placing an activity within a particular swimlane indicates that it is performed by a person or persons within the organization. Lines across swimlane boundaries indicate interactions among different organizations, which must usually be treated with more care than interactions within an organization. The horizontal arrangement of swimlanes has no inherent meaning, although there may be situations in which the order has meaning.



**Figure 8.8 Activity diagram with swimlanes.** Swimlanes can show organizational responsibility for activities.