Unit V

CHAPTER 5

Class Design, Implementation Modeling, Legacy Systems

University Prescribed Syllabus

Class Design : Overview of class design; Bridging the gap; Realizing use cases; Designing algorithms; Recursing downwards, Refactoring; Design optimization; Reification of behavior; Adjustment of inheritance; Organizing a class design; ATM example. Implementation Modeling : Overview of implementation; Fine-tuning classes; Fine-tuning generalizations; Realizing associations; Testing.

Legacy Systems : Reverse engineering; Building the class modes; Building the interaction model; Building the state model; Reverse engineering tips; Wrapping; Maintenance.

Specific Instructional Objectives :

At the end of this lesson the student will be able to :

- Identify the classes.
- Prepare data dictionary.
- Identify associations.
- Identify attributes of objects and links.
- Organize and simplify classes by means of inheritance.
- Verify access paths.
- Explain reconsideration of the level of abstraction.
- Grouping of classes into packages.
- Identify and define system boundary.
- Identify actors.
- Identify use cases.
- Define initial and final events.
- Define normal scenarios.
- Identify exceptions and variations in scenarios.
- Identify and define external events.
- Design activity diagram for use cases.
- Explain System Design.
- Describe Design Algorithms
- Describe Design Optimization
- Design Algorithms
- Design Optimization.

5.1 System Analysis

GQ. Write a short note on: Data Dictionary.

- These chapter emphases on fundamental concepts of Object Oriented System Analysis and System Design.
- Object oriented analysis methods makes use of the basic object oriented concepts in requirements analysis phase of the software development life cycle (SDLC) for analysis of the proposed software system.
- The static and dynamic archetypes of the proposed software system are designed in the system analysis phase. The main motive behind system analysis is to find out real-world objects and entities in the proposed system scenario and then plotting these objects as the software objects in the system scenario.
- The static object modelling in object oriented analysis comprises of object modelling and class modelling. The dynamic object modelling in object oriented analysis consists of state modelling, activity modelling, use case modelling and interaction modelling.
- Object modelling determines objects while class modelling determines classes to which the identified objects belong and present the relationships amongst classes and objects by means of class diagram.
- So in this chapter, we are going to study basics of system analysis phase in software development life cycle and fundamental issues of object oriented analysis and design.

5.1.1 Finding Classes

GQ. Give the detailed guidelines for finding and defining the classes involved in the software system scenario. Explain with suitable example.

- As we have already discussed, a class is a collection of objects those are having common characteristics. A class can be defined as a detailed explanation of a group of objects that share the equivalent attributes, relationships and operations.
- A class itself is not a specific object however; it is a complete set of objects.
- Typically, objects of a class are distinctive in nature and they are uniquely identified by means of variances in their attribute values and definite association to other objects of the same class. A class of a particular object is a fundamental feature of that object.
- For each object, it is possible to identify and distinguish its own class. Object oriented programming languages are capable of describing a class of an object at run time.
- Finding the appropriate classes in the proposed system scenario is quite essential in the system analysis phase of SDLC.
- For finding the classes, no stepwise procedure or algorithm is available in the literature. If we try to design such algorithm for finding out the classes, then it will be more dependable method of object oriented analysis and design. But, no such mechanism is available in the literature.
- So, analysis of classes is the whole sole responsibility of the personality known as ***software system analyst***. The experienced software system analysts may help out for defining the classes of the proposed software system in right manner.

- Sometimes, two or more classes may describe the same context then; we should choose one of them which is more relevant to the respective situation. So, better way chooses the class which is more descriptive. Also, the inappropriate classes should be removed.
- For instance, following is the possible list of classes in the case study of ATM Machine :
  - Customer/User
  - System User/Bank Employee
  - Credit/Debit Card
  - Receipt
  - ATM Machine
  - Banking Network
  - Central Server
  - Computer System at Bank
  - Bank Branch/Office
  - Customer Account Data
  - Transaction
- We can find out, define and design classes by using :
  - Noun/Verb Analysis
  - Class, Responsibilities and Collaborations (CRC) analysis and
  - Rational Unified Process (RUP) stereotypes
- Let us have a brief discussion on the said strategies for finding the classes.

5.1.1.1 Noun/Verb Analysis for Finding Classes

- For determining classes and their attributes/responsibilities, noun/verb analysis is the simplest strategy. We know that, the UML notation of a class consists of three sections: class name, class attributes and class operations.
- The class name depicts the name of an entity/actor/component/node/system involved in the system scenario. The class attributes are the characteristics or features of a particular class and are used for collection of objects having similar features and properties.
- While the class operations depicts the responsibilities of a particular class in the system archetype. In the procedure of noun/verb analysis for finding out the classes, noun phrases or nouns are dedicated for defining classes themselves or for defining attributes of respective classes and verb phrases or verbs depicts operations or responsibilities of the respective classes.
- System analysts have to be very careful in understanding the proposed system domain first and they should acquire better knowledge regarding system problems. The noun/verb analysis is the two-step procedure for finding the classes.
- First step is collection of the information and second is analysis of gathered information. In the beginning of the noun/verb analysis process, it is very essential to gather maximum system related information.
- System analysts can acquire system related information from different resources such as the proposed system vocabulary, the use case model, the requirements model, proposed system framework and many more.
- After collecting information from different data sources, analyze the information with the help of grammatical terms like noun phrases, nouns, verbs, verb phrases, etc.
- For instance, following is the categorization of the information in ATM Machine case study.
  - **Nouns :** Customer, Account, etc.
  - **Noun Phrases :** Customer ID, Account No., etc.

- **Verbs :** Open, Transfer, Manage, etc.
- **Verb Phrases :** Opening an account, Amount transfer, transaction, etc.

- In conclusion, nouns and noun phrases may specify classes or class attributes whereas verbs and verb phrases specify operations or responsibilities of classes.

5.1.1.2 CRC Analysis

- CRC stands for Class, Responsibilities and Collaborations. CRC component comprises of three partitions: class name, responsibilities and collaborations.
- The responsibilities and collaborations are positioned at the same level where, responsibilities are placed in the left side and collaborations are placed in the right side.
- Responsibilities enlist the tasks, functions or operations for which a particular class is responsible and collaborations compartment enlists the other classes that collaborate with a particular class.
- Fig. 5.1.1 shows the structure of CRC.

Class Name : Bank Account	
Responsibilities	**Collaborations**
- Initiate Transaction	- Bank ATM
- Money Transfer	- Bank

Fig. 5.1.1 : CRC component

- CRC analysis strategy should be used along with noun/verb analysis process. CRC analysis is also a two-step procedure: Collecting information and Analyzing information.

5.1.1.3 RUP Stereotypes

- RUP stands for Rational Unified Process. Concept of analysis class is considered in this process. Analysis classes are the classes that signify an abstraction in the proposed system domain and are mapped to the real-world entities.
- Different types of analysis classes are used in finding the classes using RUP stereotypes. Entity class, control class and boundary class are the types of analysis classes.
- Entity class gives determined information about the particular entity involved within the system scenario.
- Control class summarizes use case modelling and helps in describing use case model in brief.
- Boundary class is used as a facilitator between proposed system and outside environment. It provides a platform for communication and coordination between the system and its environment. Fig. 5.1.2 shows different types of analysis class and their UML notations.

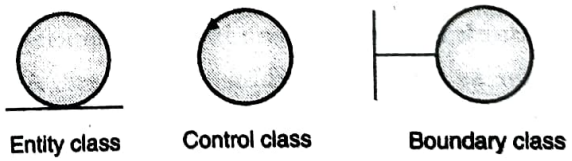


Fig. 5.1.2 : Types of Analysis class

- Entity class have a simple behaviour and are familiar with entities in the system.
- Control class acts as a controller and hence it controls the overall behaviour of the proposed system.

- Boundary class is used for defining the boundary of the proposed system. Boundary classes are furthermore categorized into three types: system interface class, user interface class and device interface class. All of these classes act as a mediator between systems, users and external devices respectively.

5.1.2 Data Dictionary Preparation

GQ. Prepare a data dictionary for ATM system scenario. Explain each element in brief.

- Data dictionary is the phrase related to 'data' and is the thing important for modelling all of the components involved in the system scenario.
- System analyst should briefly explain each class specifically in a passage of at least five statements that exactly describe the class. Also, scope of the class should be mentioned along with assumptions .
- Furthermore, the data dictionary specifies attributes, responsibilities, operations and associations. Let us have a brief discussion on data dictionary for ATM machine.
- We have already identified classes involved in the ATM machine example in Section 5.1.1. As far as the concept of data dictionary is concerned, data dictionary will define each of the class in brief.
- Once again, let us enlist the possible classes involved in the ATM machine.
  - Customer/User
  - System User/Bank Employee
  - ATM Card
  - Receipt
  - ATM Machine
  - Banking Network
  - Central Server
  - Computer System at Bank
  - Bank Branch/Office
  - Customer Account Data
  - Transaction

Now, let us define each class in brief in order to prepare data dictionary for ATM machine.

1. **Customer/User :** Customer in the ATM machine scenario is the human being and is the account holder in a particular bank. Customer/User might handle one or more accounts in the same bank. Customer/User can be an individual customer or a group of persons, organizations or corporations. The same person can have account in different banks but in that case, that particular customer is differently considered in different banks.
2. **ATM Machine :** It is the place from where customers can process their individual transactions by means of ATM card supplied by the bank for the purpose of unique identification of the customer. The ATM machine acts as a mediator between bank and customer and provides platform for handling transactions amongst customer and bank.
3. **Bank Branch/Office :** It is the financial organization that provides banking facilities to customers. As far as the ATM machine is concerned, ATM cards are important for transaction processing and these ATM cards are supplied by the respective bank only. Bank issues the ATM card to individual account holder and hence allows customer to handle the account in the bank.
4. **System User/Bank Employee :** These are the authorized and official personalities of a bank to handle the transactions initiated by a particular customer. Ultimately, these personalities are responsible for handling accounting and financial issues of an individual customer.

5. **Banking Network :** It consists of network components or nodes like ATM machines, banks central server connected to each other for communicating and coordinating the transactions initiated by individual customer. To handle the customers' transaction in real time is the whole and sole responsibility of the banking network and hence without banking network, the ATM machine services are unworkable and are of no use.
6. **Customer Account Data :** It is the detailed record of a particular customers' account and is used for handling transactions of that particular account.
7. **ATM Card :** It is the card provided by the bank for unique identification of a customer and transactions handled by a customer. ATM card comes with a card number, bank code on it and a PIN. The card number identifies a particular account while the bank code uniquely identifies the bank.
8. **Central Server :** It is the computer machine that dispatches transactions amongst the computer systems in the bank and ATM machines which are geographically distributed at different locations.
9. **Transaction :** It is a solitary request for performing operations on accounts of customers.
10. **Receipt:** It is the machine generated token and depicts record of transaction performed by a customer.
11. **Computer System at Bank :** It is the computer system used by bank employees like cashier, manager, etc. through which these employees can enter transactions for customers. These computer systems communicates and coordinates with the central bank server to authenticate and process the transactions.

5.1.3 Association Design : Finding Associations

- Association is a structural relationship amongst two classes and is static in nature. It shows fixed relationships between classes involved in a system.
- Association is shown by a solid line between two classes and it can be called as a binary association.
- Moreover, a reference from one class to the other class is an association.
- Associations can be mentioned in terms of verb phrases or verbs.
- Verb phrases used in communication, for specifying physical locations, for describing some conditions and many more can be used for showing association between two classes.
- In case of an association relation, the similar objects can form a group but all of the objects in a particular group are not totally dependent on each other. Therefore, association is considered as a weak form connection in object oriented programming.
- For example, consider a bus, number of passengers and a driver. We can say that, the passengers inside the bus and the driver are associated when they are in the bus because all of them occupy some space inside the bus and all of them move in same direction.
- A structural property of the problem should be described by an association.

5.1.4 Finding Attributes of Objects and Links

- An object itself is an instance of a class which outlines the set of characteristics or features that are unitedly used by all of the instances of the respective class.
- We can think of an object as a unified pack of data and function. The data is nothing but the information about an object and functions are known as ***operations***.
- An object have its own qualities and characteristics known as ***attributes***. For instance, a person has a name, a qualification, a color, an age and hobbies; a vehicle has a number, a manufacturer, a color, a shape, an owner and a price. The attribute values hold an object's information.
- Objects also can have behaviour. For instance, a vehicle can move from one location to other. Also, objects have their specific self-determining attributes. These attributes plays a vital role in unique identification of an object in a particular scenario.
- Attributes of an object depicts the state of an object. For example, the properties of a motorbike can be manufacturer, color, cost, number of gears, etc., and are the attributes of a motorbike object (Fig. 5.1.3).

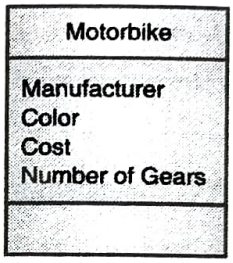


Fig. 5.1.3 : The attributes of a motorbike object

- Each attribute of an object can be represented in different ways in a programming language. For instance, manufacturer of a bicycle can be signified by a name of a manufacturer, unique commercial tax identification number or a reference to a manufacturer object.
- A link is defined as a semantic connectivity amongst two objects and it supports message passing mechanism for transfer of data from one object to other object since, objects requires communication between each other in an object oriented program.
- A link is a physical connectivity between two objects while an association is a physical connectivity between two classes. So, we can say that; link in object diagram is replaced by an association in class diagram and both are having same motive as far as their functionality is concerned. Moreover, a link is an instance of an association.
- Normally, most of the links are used for representing connectivity between two objects but it is quite possible to relate three or more objects with a single link. The links of an association relationship syndicates objects from the similar classes.

5.1.5 Organizing and Simplifying Classes using Inheritance

- Number of classes can be clustered into a single group and a class can inherit some of its characteristics from a parent class.
- Inheritance is the mechanism with the help of which specific elements can obtain their behaviour and structure from general elements involved within the system scenario.
- For organizing and simplifying classes with the help of inheritance, system analyst should first of all look after responsibilities, operations, and attributes that are common to two or more classes in a group of classes.

- Furthermore, system analysts should promote these common attributes, operations and responsibilities to a general class. System analyst can design new class for promoting these things but if possible, it is better to avoid inclusion of new class and system analyst should incorporate the necessary modifications in the existing classes only.
- It is possible for a class to have more than one super class in Unified Modelling Language. That means, a child can have more than one parent and this property is known as Multiple Inheritance.
- Multiple inheritances is not supported by all of the object oriented languages. C# and Java permits only single inheritance. A child class can inherit properties of its super classes in multiple inheritances.
- Implementation of multiple inheritances is possible in object oriented programming only if it is supported by the target implementation language. This can be considered as one of the design issue in object oriented programming and hence, implementation of multiple inheritances is totally dependent on the target object oriented language.
- The "***is kind of*** "relationship should be applied between the super classes and subclasses. Normally, super classes should not have a parent class in common in order to avoid cycles in the inheritance hierarchy.
- All the super classes involved in the scenario should be semantically disjoint. Sometimes, classes are defined to be "mixed in" with other classes with the help of the inheritance. Such types of classes are known as ***mixin classes***. Mixin classes supports multiple inheritances effectively.
- The multiple inheritances allows private inheritance and is powerful as compared to the single inheritance. Also, multiple inheritance permits mix-in inheritance. Repeated inheritance, clashes in names and complexity are some of the drawbacks of a multiple inheritance.
- In conclusion, inheritance is natural, sophisticated and graceful in nature and we can write generic code by making use of the concept of inheritance.
- But, it may not perform well during execution of the system and sometimes it is more complicated for better understanding.
- Furthermore, we can determine inheritance from the bottom up by searching for classes with similar operations, associations and attributes and this strategy is known as ***bottom-up generalization***. In this mechanism, system modellers should refine some classes or their respective attributes in order to fit them in a superclass and we should define a super class for each generalization in order to share the similar features.

5.1.6 Verification of Access Paths

- It is moderately essential to test and verify each access path in the system scenario. Software modellers should verify that, a particular access path produces fruitful outcome as per the requirement given by the end user.
- We can have independent classes in the system scenario for performing unique tasks and operations.
- Sometimes, these independent classes are not connected to any of the class involved in the class model and are disconnected in nature.

5.1.7 Reconsidering the Level of Abstraction

GQ. What is reconsideration of level of abstraction? List and explain several categories of abstraction.

- Abstraction is the fundamental way in which the definite facets of a particular problem are scrutinized selectively. The main objective of an abstraction is to segregate those facets of a particular problem which are essential for some purpose and destroy those features which are irrelevant.
- Abstraction should be purpose specific, for the reason that the purpose defines what is important, and what is not important. An abstraction signifies the crucial features of an object that differentiate it from all other types of objects.
- All abstractions are partial (incomplete) and erroneous (incorrect). An abstraction mainly concentrates on the exterior view of an object and assists for separation of an object's important behaviour from its implementation.
- The main motive behind an abstraction is to limit the universe so we can recognize and realize. Abstraction can be categorized into four types :
  - ***Entity abstraction :*** An object which denotes a useful model of a problem-domain entity or solution-domain entity.
  - ***Coincidental abstraction :*** An object that bundles a set of operations which are not interrelated with each other.
  - ***Virtual Machine Abstraction :*** An object that makes clusters of operations which are used by some superior level of control or operations that make use of secondary level set of operations.
  - ***Action abstraction :*** An object that offers a general set of operations, all of which implement the similar type of function.
- All abstraction has static as well as dynamic properties.

5.1.8 Grouping of Classes into Packages (Physical Packaging)

GQ. Give the guidelines for grouping of classes into packages.
GQ. List and explain types of dependencies in package.

- A class can be defined as a detailed explanation of a group of objects that share the equivalent attributes, relationships and operations.
- A class itself is not a specific object however, it is a complete set of objects.
- An object is characteristically an occurrence of a class. A class depicts a set of objects having similar attributes (properties), meanings, operations (functionality) and types of relationships.
- For example; employee, bicycle, fruit, college are the classes. Every employee working in a particular organization has employee-id, name, department, post, qualification, etc. as attributes that can be used for unique identification of a particular employee in an organization. As discussed in section 5.1.4 of this chapter, motorbike has characteristics like manufacturer, colour, cost, number of gears, etc. Each fruit has name, colour, shape and flavour. College has its own attributes as name, affiliation, type etc.
- When we have to extend the system to some more extent, it is quite necessary to organize all of these contents collectively into large containers. The package is defined as a container that organizes a model by grouping things. Hence, packages helps us to arrange all of the elements collectively for better understanding of the system archetype.

- The package is a widespread mechanism dedicated for organizing all of the building blocks of UML into groups. Normally, packages are used for presenting behavioural and structural views of the software system.
- Semantically associated elements are grouped by means of concept of package. We can have nested packages inside other packages.
- By rule, there should not be cycles in dependency relationships amongst packages involved in the scenario.
- The interfaces of the packages should be stable in case of the numerous dependencies in between packages.
- There are five categories of dependencies amongst packages.
  - ***<<import>>*** : The <<import>> dependency combine client and supplier namespaces and it implements a public merge.
  - ***<<access>>*** : The <<access>> dependency also combine client and supplier namespaces. Only the difference is that, it implements a private merge.
  - ***<<use>>*** : The <<use>> dependency represents the interrelationship between the elements within the packages and not the packages themselves.
  - ***<<merge>>*** : The <<merge>> dependency is used in met modeling only. We should not take into consider this kind of dependency in case of object oriented modeling and design.
  - ***<<trace>>*** : The <<trace>> dependency typically shows the interrelationship between models (archetypes) instead of depicting the relationship between elements of a system.

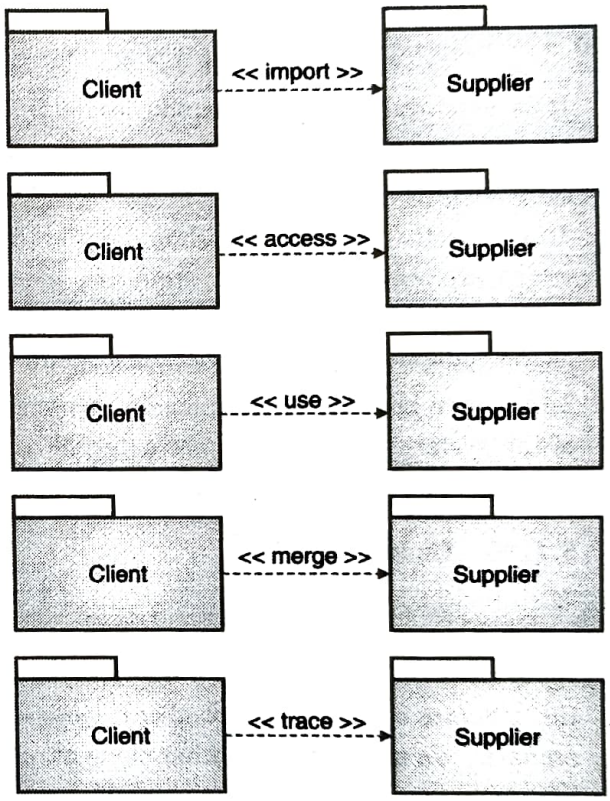


Fig. 5.1.4 : Categories of package dependencies

- We can merge classes from one package into another package by means of dependency. Package diagrams should be used in case of larger systems in order to get a detailed representation of the dependencies amongst key elements of a system.
- Dependencies within a proposed system can be handled effectively with the help of package diagram.
- For example, the detailed model of an ATM Machine can have following packages :
  - **Bank Employee :** Manager, Cashier, ATM Manager.
  - **Bank :** Branch, ATM.
  - **Account :** Type, Customer, Credit/Debit Card, Transaction Processing, etc.

Guidelines for Grouping of Classes into Packages

- A well-organized package is loosely coupled and normally less nested.
- Packages should not be too large for better understanding of the system archetype.
- Simple form of the package (i.e. folder) should be used for representation of a package. Detailed contents of the package (set of classes) should be given on the body of the package in extreme cases only.
- Only meaningful contents (objects, classes, relationships and interfaces) which are important to understand the system model should be shown explicitly.

5.1.9 Determining System Boundary

GQ. Write a short note on : Determining System Boundary.

- It is essential to define boundaries of the system before moving towards the implementation and execution of the system. For determining the system boundary; first of all, software designers should know the specific scope of the proposed software system.
- That is, software designers should define the things which are external to the system and the things which are the part of the system. So, the things which are the part of the system comes inside the system boundary which is also known as a subject while the things which are external to the system are located outside the boundary of the system.
- The system boundary is graphically represented by means of a rectangular box, labelled with the name of the system. Since, actors are external to the system; they are drawn outside the system boundary. Use cases are drawn inside the system boundary.

5.1.10 Finding Actors

GQ. Explain the term finding actors with respect to use case diagram.

- An actor is the external user of the system who is responsible for communication and coordination with the software system. It is not always necessary to have a human being as an actor within the use case scenario. We may have any other element or an external system as an actor.
- It is first of all necessary to recognize the role of an actor for better understanding of an actor. For finding actors of a particular system, following questions should be taken into consideration:
  - Who are the end users of the system?
  - Who are the installers of the system?
  - Who provides information to the system?
  - Is there any other cooperating or interacting system available in the scenario?
  - Who maintains the system?
- From the business point of view, each actor should be named uniquely. Always, there is direct communication and coordination amongst actors and the system.

- We may have time as an actor in case we have to model the things that happen at a particular point of time in the system scenario. Actors are always external to the system.
- There is direct interaction between actors and the system; i.e., actors can directly communicate and coordinate with the system.
- Each actor should be described in short (in one or two sentences) for better understanding of the exact role of an actor within the system.

5.1.11 Finding Use Cases

GQ. Explain the term finding use cases with respect to use case diagram.

- In UML, the system requirements and functionality of the system are depicted with the help of use cases.
- Use cases are meant for specification of the interaction between the system itself and end users of the system which are termed as actors in UML.
- Basically, use case offers a detailed description of how the system is used. The set of activities and events in some proper sequence specifying the interaction amongst a system and its end users is known as a scenario.
- We might have to handle several scenarios in the execution of the system depending on the situations or scenarios involved within the proper execution of the system.
- For instance, in case of an ATM machine, access will be given to authorize customer only by authenticating the particular customer through his/her card and PIN. If login is successful, then customer will be in position to perform the transaction. But, if the login is failed, no access will be given to the customer and customer will not be able to perform the transaction through an ATM machine. So, both of these scenarios are different which are the things that might happen.
- All of the scenarios mentioned in the above example are different but are equivalent since the customer has the same goal in all of the scenarios i.e., to perform the transaction. This goal is only the main component behind definition of use cases.
- A use case is defined as a set of scenarios that collectively work to achieve a common user goal. It outlines a sequence of interactions amongst one or more actors and the system itself.
- In the above example, transaction processing can be one of the use case that strives for the successful transaction processing that is the goal of the customer.
- Each use case comes with its primary actor who is responsible for certain tasks involved in the scenario. More details of the actor are discussed in subsequent section of this chapter.
- Each use case should clearly mention the exact interaction between the actors and the system itself.
- Furthermore, use case guides us about what the system exactly does in response to the actor's activity and it is not devoted for detailing how system does it.
- At all times, a use case starts with input from a respective actor. Thus, an actor is responsible for giving input the system and the system is dedicated for giving response to the actor.
- A simple use case encompasses a single interaction between the actor and the system. But, a single use case can handle set of interactions between a particular actor and the system. More than one actor can be a part of the complex use cases.

- Graphically, a use case is shown with the help of an oval shape containing the use case name inside its body. Each use case should be named uniquely and use case name should describe the desired functionality of that particular use case.

5.1.12 Finding Initial and Final Events

GQ. Define event. Define and describe initial and final events for ATM system.

- Things that occur at a particular instance of time are known as events. Every single thing or action occurred at a particular time instance is termed as an event in UML.
- An event owns an activity or operation along with its time and location. As far as use case model is concerned, the proposed software system can be divided into small pieces on the basis of use cases.
- In use case model, use cases specifies the flow of execution of the system along with respective actors but it do not depict behaviour. So, events are used in order to represent the behaviour.
- The order of execution of events is better for understanding the behaviour of the system. Each use case involved in the scenario should be covered by means of sequence of events.
- From the use case model, we can define initial and final events by means of actors and use cases defined. Normally, each use case is dedicated for provision of some kind of service in the system scenario for proper execution of the software system.
- Defining initial events is simply nothing but the demand of service from respective use cases during execution.
- Sometimes, initial event causes execution of several activities in proper sequence that leads to fruitful outcome from the system. Same as that of the initial event, it is quite essential to define final event too.
- The final event simply determines the boundary line for a particular activity or set of activities in some sequence. i.e.; it tells us where to stop or where to terminate. Let us define initial and final events for ATM example.
- Main activities involved in the ATM are: Transaction initiation, Data transmission and Transaction processing.
- In case of transaction initiation, the initial event can be insertion of ATM card into the provided slot in the ATM machine. Final events can be either ATM machine holds ATM card in the slot till the end of the transaction or ATM returns the ATM card if PIN entered by the customer is invalid.
- During data transmission, initial event is customer requests account data and hence customer initiates the transaction while the final event is ATM machine communicates with the bank server, retrieves customers' account data and displays the same on ATM screen.
- During transaction processing, initial event is actual transaction initiation and final event can be transaction completion or transaction termination.

5.1.13 Preparing Normal Scenarios

GQ. What do you mean scenario. What is the purpose of defining the scenario? Explain with suitable example.

- A scenario is nothing but the ordered set of events or activities that occurs in between objects and it illustrates behaviour. Scenario can be defined as a definite path through a use case.

- As discussed earlier, when internal objects communicates and coordinates with the external objects, an event occurs. So, certain message interaction amongst objects in the interaction model is considered as a scenario and it is not a use case. Each use case may comprise of several scenarios in case of a complex system.
- Crucial or necessary sequences involved in a particular use case are considered as primary scenarios and alternative or substitute sequences are considered as secondary scenarios for each use case in the system archetype.
- The data values involved in the activity are termed as event parameters. For example, in "enter username and password" activity; username and password are event parameters and these event parameters are exchanged amongst internal elements of the system and external agents.

Let us discuss scenarios for some of the use cases involved in the ATM machine example.

- Scenario for transaction initiation comprises of following sub tasks :
  1. The ATM machine shows Welcome message on the screen.
  2. The ATM requests the customer to insert an ATM card into the slot.
  3. The customer inserts an ATM card.
  4. The ATM machine accepts the card and scanner inside the ATM machine scans and reads the serial number of an ATM card.
  5. The ATM machine asks customer to enter the valid PIN.
  6. The customer enters valid PIN.
  7. The ATM machine accepts the PIN and communicates the same with the central server of the bank in order to validate the customer.
  8. After validation, the ATM machine displays the main menu with the help of which the customer can initiate, process and terminate the transaction.
- Data transmission scenario consists of :
  1. The ATM machine accepts input from the customer.
  2. The ATM machine requests customers' bank account data from the banks' central server.
  3. The respective bank's server receives the request of account data from the customer.
  4. The bank server retrieves the account data as per the customer's requirement/query.
  5. The bank server then transfers the account data to the ATM machine.
- Actual transaction processing scenario comprises of :
  1. The ATM machine shows a main menu for transaction processing.
  2. Let us say, the customer selects money/cash withdrawal option on the ATM machine screen.
  3. The ATM machine asks for the amount of cash to be withdrawn.
  4. The customer will then enter amount of cash; for e.g., 1000 Rs.
  5. The ATM machine then confirms policy limits for cash withdrawal.
  6. The ATM machine connects to the banks' central server and verifies that, the particular customers' bank account has adequate cash.
  7. The ATM machine dispenses the cash and requests the customer to collect it.
  8. The customer then collects the cash and gets the printed receipt of transaction.

- Subsequently, we have seen three use cases from the ATM machine which are : transaction initiation, data transmission, actual transaction processing and defined their scenarios.

5.1.14 Adding Variation and Exception Scenarios

- After defining normal scenarios, software system designers should give focus on special cases involved in the system archetype.
- The special cases contained in the system archetype can be lowest or extreme values, repetitive values, erroneous input or misplaced input and many more.
- In addition to that, error cases like illegal values, failures at particular stage, etc. should also be considered.
- For instance, the ATM machine may have following exceptions and variations for which we can define scenarios :
  - Card Read Error: The ATM card is not readable.
  - The ATM card is damaged and is not working.
  - The amount entered by the customer is invalid.
  - The ATM machine times out and waiting for response.
  - The ATM machine is out of cash.
  - The ATM machine is down/not working.
- Let us have a look at the scenario for one of the exception: Card read error.
  - The ATM machine shows Welcome message on the screen.
  - The ATM requests the customer to insert an ATM card into the slot.
  - The customer inserts an ATM card.
  - The ATM machine accepts the card and scanner inside the ATM machine scans and reads the serial number of an ATM card.
  - If the ATM card is damaged or is not working, the card read is displayed on the screen.
- In the same manner, we can write scenarios for remaining exceptions and variations mentioned above. For practice, students should try to write scenarios for the above exceptions and variations.

5.1.15 Finding External Events

- Previously, we have already discussed how to find initial and final events.
- Now, we have to define external events. Software modellers should start specification of external events with the help of scenarios defined.
- The scenarios will acts as a baseline for defining the external events.
- All types of inputs to the software system, possible interrupts, decisions and interactions between end users and external devices are useful for defining the external events.
- For better understanding, software modellers should design sequence diagram for each of the scenario.
- A sequence diagram should be used when we have to depict the flow of control within a system by time ordering of messages amongst objects.

- A sequence diagram should contain only important objects, nodes, components, collaborations or actors.
- In short, the sequence diagram evidently illustrates the sender and receivers of each event.
- Fig. 5.1.5 shows the sequence diagram for actual transaction processing which is one of the scenarios that we have discussed in earlier sections of this chapter.

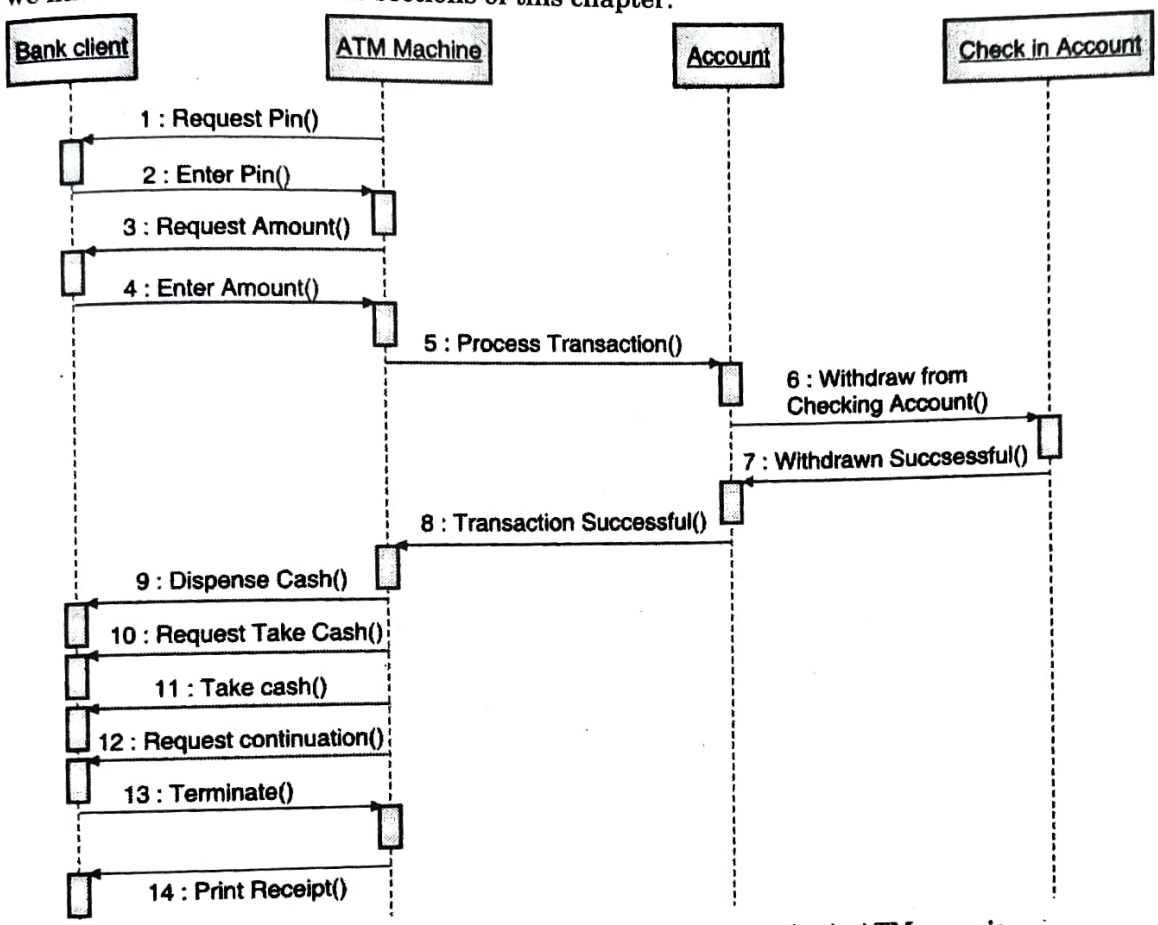


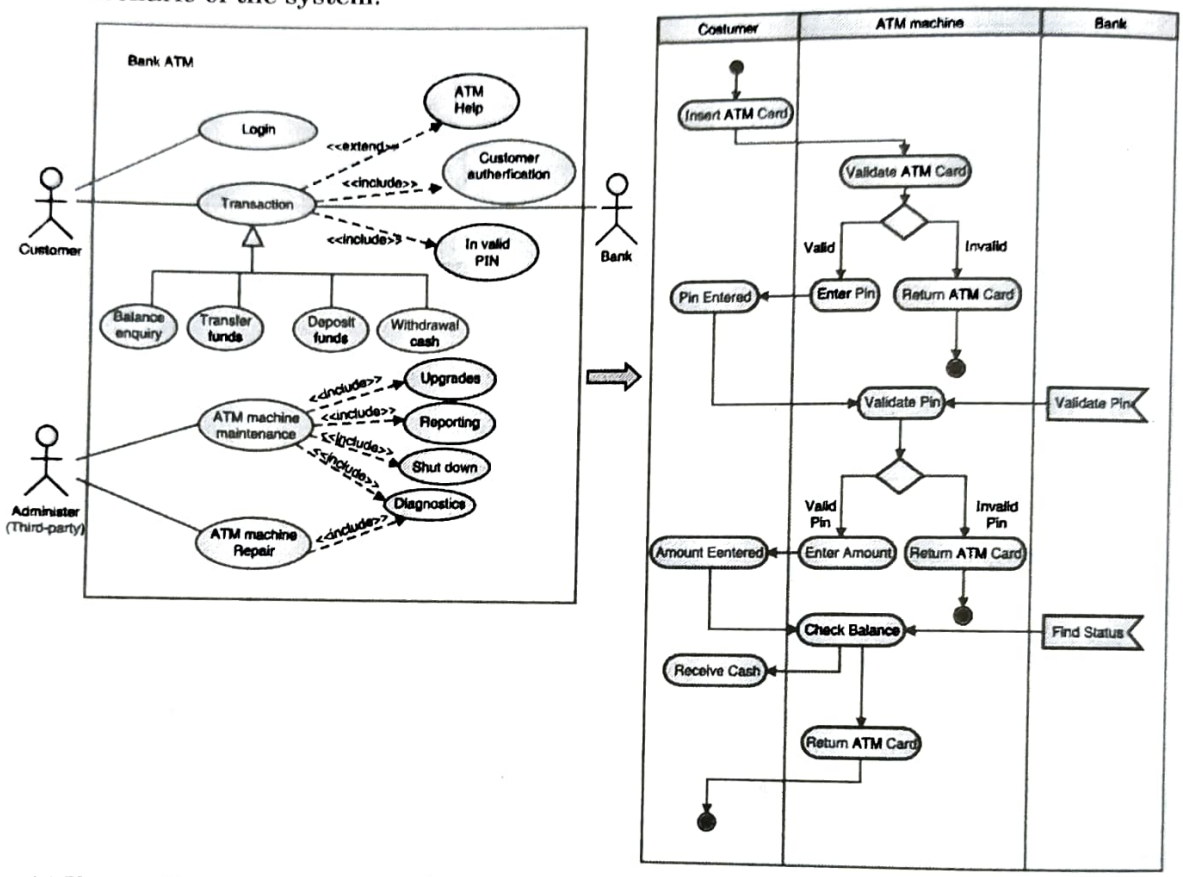
Fig. 5.1.5 : Sequence diagram for actual transaction processing in ATM scenario.

5.1.16 Combining Models : Preparing Activity Diagram for Use Cases

GQ. Explain in detail the process of preparing an activity diagram from use case diagram. Give suitable example.

- An interaction diagram comprises of set of objects, their relationships and messages sent amongst them and are mainly devoted for showing interaction between the objects.
- An interaction diagram is meant for showing communication and coordinate recursing on between two or more objects but, data manipulation is not shown. The interaction diagrams typically comprises of sequence diagram and collaboration diagram.
- The sequence model or sequence diagram is a type of interaction diagram that primarily focuses on time ordering of messages. The objects in the sequence diagram are not only the instances of class but they can be instances of elements like nodes, components and collaborations involved in the scenario.
- The sequence diagram uses the object timeline in order to specify the time ordering of the messages between objects. But, decision points and alternatives are not clearly shown in sequence diagram.

- That is, software modellers have to draw individual sequence diagram for each of the scenario involved in the system archetype. For instance, actual flow of interaction within the system will need separate sequence diagram and additionally we will need several sequence diagrams for decision points and errors.
- In order to tackle this problem, activity diagram helps us for representing step by step flow of execution of the system by considering the system as a whole.
- Activity diagram focuses on flow of control from activity to activity and hence it considers entire activity as a whole. The components like merges, forks and control flow helps us to depict overall step by step execution scenario of the system.



(a) Use case diagram for Overall ATM machine **(b) Activity Diagram for overall ATM machine**

Fig. 5.1.6 : Activity diagram from Use case diagram

- An activity diagram can be used to provide a detailed description of the use case. One or more sequential steps of a particular use case can be represented by means of an activity node.
- Normally, a high level activity node is used to show a single use case and then it can be separated into a separate activity diagram. Correspondingly, an activity diagram is a good option for showing proper ordering between use cases involved in the system scenario.
- So, use cases in use case diagram are replaced by activity nodes in the activity diagram, relations like association, generalization, include and extend in use case diagram are replaced by loops, arcs and decision points depending on the situation in the system scenario. An activity node depicting a use case can also be used for showing a link to an inclusion use case or exclusion use case.

- After decomposition of a high level activity node, we can get low level activity diagram showing precise flow of a particular activity. An example of use case diagram and activity diagram for overall ATM machine is shown in Fig. 5.1.6.

5.2 System Design : Organizing a System into Subsystems

GQ. What do you mean by System Design? Illustrate with suitable example.
GQ. How can we organize a system into subsystems? Explain in detail.

- The very first step in software system design is to divide the system into number of pieces for better understanding of the software system scenario. Each key fragment of the system is known as a ***subsystem.***
- Each subsystem is based on some common theme, such as similar functionality, the same physical location, or execution on the equivalent kind of hardware.
- For instance, a spacecraft computer system might contain subsystems for life support, navigation, engine control, running scientific experiments and other activities.
- A subsystem is a group of classes, associations, operations, events and constraints that are unified, incorporated and have well-structured and defined small interface with other subsystems.
- A subsystem is typically acknowledged and approved by the numerous services provided by it.
- A service is a cluster of related functions that share some mutual purpose, such as I/O processing, drawing pictures or performing arithmetic.
- A subsystem articulates a rational technique of looking at a piece of the problem. For instance, the file system within an operating system is a subsystem that encompasses a set of associated abstractions that are generally independent of abstractions in other subsystems such as process management and memory management.
- Each subsystem has well-structured and defined interface to the rest of the system.
- Layers or Partitions can be used in order to divide the complete software system into subsystems in horizontal or vertical manner respectively.
- That is, software designers can organize the subsystems horizontally by means of layers and vertically by means of partitions.
- In both of these cases, all layers or partitions involved within a software system are interrelated with each other and interrelationships amongst these layers or partitions is of the type client server relationship.
- The key difference between partitions and layers is that, the subsystems defined with the help of partitions will have identical and same levels of abstraction whereas the subsystems defined by means of layers will have different levels of abstraction.
- One more difference is partitions are having peer to peer relationship amongst them and layers are having client server relationship in between them. Consequently, each partition can be considered as a self-governing peer and each layer is dependent on some other layer or group of layers.
- Furthermore, we can combine partitions and layers in order to get a software system as a whole. In conclusion, we can layer the partitions and partition the layers.

5.2.1 Identifying Inherent Concurrency in a Problem

- The state model is dedicated for detailed description of sequence of tasks and services involved within the system operations and do not deals with the contents like details of operations and services, how different services and operations are executed during implementation and execution of the proposed system, etc.
- The state model describes those aspects of objects concerned with time and the sequencing of operations. When two objects can get the events simultaneously deprived of interfacing, they can be said inherently concurrent.
- As far as implementation is concerned, all of the software objects are not concurrent, since one processor can support many objects. The state model is basically articulated by means of a state diagram.
- So, with the help of a state diagram, several objects involved in a scenario can be congregated into a solitary thread of control. A thread of a control is an end to end path through a set of state diagrams of discrete objects and only single object can be active at a particular instance of time.
- For instance, in case of an ATM system, the ATM machine is idle; when the bank is authenticating and confirming an account or processing bank transaction. If the central computer system proximately controls the ATM, the bank transaction object and an ATM object can be associated with each other as a solitary task.

5.2.2 Allocation of Subsystems to Hardware

- The necessity of advanced or superior performance basically forms the basis for the decision to use various processors or hardware functional units. The number of essential processors required totally depends on the speed of the machine and the volume of calculations.
- For instance, a parallel computing system produces too much data in a very short time span in order to handle in a solitary central processing unit. Many parallel machines should abstract the information or data handled within a system well before analysing a threat.
- The software system designer should assess and calculate the mandatory CPU processing power by means of computing the stable state load as the product of the number of transactions per second and the time required to process the particular transaction. The estimate will ordinarily be inaccurate or imprecise.
- The estimate should be increased in order to tolerate the transient effects due to random variations in load in addition to the synchronized bursts of activity.
- The sum of surplus capacity required depends on the adequate rate of failure due to inadequate resources. In case of an ATM system example, the ATM machine is mainly responsible for actual online transaction processing and providing a user interface to customer for communication and coordination amongst customer and ATM system.
- A solitary central processing unit is sufficient for an individual ATM machine. The main server of a bank can be considered as a routing machine fundamentally since it receives ATM requests and communicates them to proper bank computer situated at a particular branch of a bank.

- Sometimes, a large network may comprise of multiple processors in case of an ATM system scenario. The computer systems situated at the bank branch accomplish data processing and encompass database applications.
- Each device is an object that operates synchronously along with other objects these other objects may be hardware units or devices involved in a system.
- The software system developers should decide and agree upon the fact that, which subsystems will be implemented in software and which will be executed in hardware.
- Two reasons behind implementing subsystems in hardware are cost and performance. Much of the trouble of designing a software system comes from accomplishing externally enforced software and hardware constraints.
- Object oriented design affords no magic solution however the external packages can be demonstrated as objects. The software system designers should take into account the cost, compatibility and performance issues.
- Correspondingly, the system designers must think about flexibility for future modifications or alterations. The future modifications can be future software product versioning or improvements and software product design variations. In case of an ATM system example, no such performance issues are occurred.

5.2.3 Data Storage Management

GQ. Explain data storage management in software modeling.

- Number of substitutes is available for managing the data storage that we can use distinctly or in combination. For instance, databases, files, data structures, etc.
- Different types of data stores offer trade-offs between cost, access time, capacity, cost, reliability and access time.
- For instance, a personal computer system application can make use of files and data structures as per its requirement, an accounting can use a database to connect subsystems.
- Files are simple, modest, permanent and inexpensive.
- Implementations for sequential files are ordinarily standard however storage formats and commands for random-access files and indexed files fluctuate.
- Following kind of data is appropriate and can be suitable for files:
  - Sequentially accessed data.
  - Data that can be wholly read into memory.
  - Data with low data density.
  - Uncertain data with modest structure.
  - Data with high volume.
- Databases are the other type of data store which are typically managed, monitored and controlled by means of database management systems.
- Several types of database management systems like relational databases and object oriented databases are available.

- Following kind of data is appropriate and can be suitable for databases :
  - Larger amount of data that should be handled proficiently and skillfully.
  - Data that synchronized the updates through transactions.
  - Data that needs updates at satisfactory levels of detail by several users.
  - Data that must be accessed by numerous application programs.
  - Data that should be protected against malicious access.
  - Data that is long-lasting and highly valuable to an organization.
- The software system designers must consider object oriented database management systems merely for specialty domain applications that have a wide variety of data types or that must access low level data management primitives.
- These software system applications can be multimedia applications, embedded system applications, different engineering applications and many more.
- The software system designers should make use of relational database management systems for the software system applications that requires databases since the features of relational database management systems are adequate and satisfactory for utmost software system applications.
- Furthermore, if relational database management systems are used appropriately, they can offer a super execution 、 an object oriented archetype.

5.2.4 Global Resources Handling

GQ. Explain in brief : Global Resource Handling.

- The software system designer should recognize the global resources.
- The categories of global resources are mentioned in Table 5.2.1 :

Table 5.2.1: Types of global resources

Types of Global Resources	Examples
Physical Units	Processors, tape drives, communication satellites.
Space	A workstation screen, the buttons on a mouse.
Logical Names	Object IDs, filenames, class names.
Access to Shared Data	Databases.

- A physical unit like processor when considered as a resource, it can regulate and control overall activities involved in the scenario on its own. A global resource may also be segregated for self-governing control.
- The buttons on a mouse, a workstation screen can be the global resources of the type space. The entities dedicated for unique identification of a particular object in a given scenario can be deliberated as resources. For instance, class names in class model, object IDs or object names in object model, file names in file management system are the logical entities that are used as resources.

- In case of an ATM machine system, the account numbers of customers and the bank codes are global resources. Bank codes should be unique within the context of a bank.

5.2.5 Control Implementation : Choosing a Software Control Strategy

GQ. Explain : (a) External control (b) Internal control

- Basically, there are two types of control in a software system:

1. External control 2. Internal control

- The flows of the events between the objects involved in the software system scenario which is visible from outside are termed as **external control flows.**
- However, the control flow comprised by a process is known as an **internal control flow.**

5.2.5.1 External Control

GQ. What are the different categories of external control? Explain in brief.

- Moreover, there are three types of control for external events:

1. Procedure-driven Control 2. Event-driven Control 3. Concurrent Control

- The choice of control flow totally depends on the existing resources involved in the software system application.

1) Procedure-driven Control

- In a procedure-driven sequential system, control exists within the software program coding.
- The procedure driven control is quite easy to implement by means of conventional languages. It is the honest responsibility of a software system designer to translate events into operations among objects.
- The drawback of procedure driven control is, concurrency inherency is significant amongst objects involved in a scenario.

2) Event-driven Control

- An event-driven control is associated with the circumstances where the measurement method is inherently event-based in nature.
- Event-driven control offers adaptable and compliant control.
- From implementation point of view, it is more challenging to implement as compared to the procedure-driven control.
- As far as modularity of a software system is concerned, event-driven control flow supports for additional modularity for breaking of a software system into subsystems.

3) Concurrent Control

- Concurrent control guarantees that, the respective transactions involved within a scenario are accomplished and executed concurrently devoid of violating the integrity of data and hence, data remains whole, complete and uninterrupted within a scenario. It is also known as yes-no control or screening.

- Ultimately, there are three basic classes of concurrent control mechanisms : pessimistic, semi-optimistic and optimistic.
- Pessimistic concurrent control refers to the blocking of a transaction if that particular transaction violates the rules. Semi-optimistic concurrent control blocks transactions in some circumstances that may cause violation and optimistic concurrent control do not blocks the transaction but it postpones the respective transaction.

5.2.5.2 Internal Control

- Throughout the software system design, the software system developer expands operations on objects into lower-level operations on the same or other objects.
- Software system designers can make use of identical system execution mechanisms for developing internal objects, external objects and communication and coordination amongst internal and external objects.

5.2.6 Handling Boundary Conditions

- Three issues should be deliberated while handling boundary conditions which are initialization, termination and failure.
- At the outset, the software system should initialize parameters, constants and variables.
- Termination is generally modest as compared to the initialization since many internal objects can merely be unrestricted. A particular executing transaction or task should release the resources.
- Failure is the unintended closure of a software system. Usually, software development team members recognize that, a proposed developed software system is not working as per the requirements mentioned in the software requirements specification.
- The failures are candidly observed by software testers during thorough and systematic software testing of a software system. Misbehaviour in the execution of a software system can be considered as an indication of the failure.

5.2.7 Setting Trade-off Priorities

- The trade-off priorities should be established for the good software system design. It is the responsibility of software system designer to set trade-off priorities for a software system. The task of designing trade-off priorities encompasses several types of software development techniques.
- If customer needs a delivery of a particular software module earlier than the outstanding software modules then customer should sacrifice the overall functionality of the software system as a whole.
- It is the duty of the software system designer to define the comparative significance of the several criteria as a guide for creation of design trade-offs.
- The software system designer does not form all the adjustments but launches the priorities for constructing respective software system arrangements.

5.2.8 Selecting an Architectural Style

GQ. Write a short note on: Architectural styles.

- Numbers of architectural styles are commonly used in existing software systems.
- Following are some types of architectural styles :

1. Batch Transformation	2. Continuous Transformation	3. Interactive Interface
4. Dynamic Simulation	5. Real Time System	6. Transaction Manager

5.2.8.1 Batch Transformation

GQ. What do you mean by batch transformation?

- In batch transformation, the information transformation is executed once on a complete input dataset. This architectural style accomplishes sequential computations.
- The main objective is to calculate an answer and is achieved by the application which is meant for receiving the inputs. Stress analysis of a bridge, payroll processing are the classic applications of batch transformation.
- In batch transformation, software developers should first of all breakdown the complete transformation into stages with each stage accomplishing one part of the particular transformation which is then followed by Formulation of class models (class diagram) for the input, output and in between each pair of successive stages.
- Next step involved is, expansion of each phase or level until the operations are straightforward to implement and finally reorganize the ultimate pipeline for optimization.

5.2.8.2 Continuous Transformation

GQ. What do you mean by continuous transformation?

- It is a system in which the output of the system is aggressively dependent on varying inputs. An uninterrupted transformation updates system outputs frequently. Windowing systems, signal processing are the classic applications of continuous transformation.
- In continuous transformation, the first step involved is to breakdown the complete transformation into stages with each stage accomplishing one part of the transformation. Then, we should describe and summarize input, output and intermediate models amongst all of the pairs of successive stages, as for the batch transformation.
- After successful breakdown of the complete transformation and defining inputs and outputs, we should discriminate each operation in order to update the incremental modifications or alterations to each level. Finally, we have to add transitional objects for optimization purpose.

5.2.8.3 Interactive Interface

GQ. Explain : Interactive Interface

- It is a system that is conquered by interactions amongst the external agents and the system itself. The external agents can be devices or humans. These external agents are self-governing and are independent of the system, so the system cannot control the agents.

- While using an interactive interface as an architectural style, we should first of all Segregate interface classes from the application classes.
- Predefined classes should be used for communication and coordination amongst the external agents. Next, we should separate out the logical events from physical events. Logical events correspond to multiple physical events. At last, we should identify and state the application functions that are invoked by the interface.

5.2.8.4 Dynamic Simulation

GQ. Explain : Dynamic simulation

- This architectural style is dedicated for designing and modeling real world objects. Video games can be the classic examples of this type.
- The internal objects in the dynamic simulation correspond to real world objects and hence the class model (class diagram) is ordinarily significant.
- While selecting a dynamic simulation as an architectural style, we should diagnose active real-world objects along with the discrete events that relates to discrete interactions with the object from the class model (class diagram). Constant dependencies should also be predicted.

5.2.8.5 Real Time System

GQ. Explain : Real time system

- The real time system is an interactive system with close-fitting or tight time constraints on actions.
- Real time system design is multifaceted and encompasses issues like interrupt handling, coordination of multiple central processing units, etc.

5.2.8.6 Transaction Manager

GQ. Explain : Transaction manager

- It is a system dedicated for retrieval and storage of data. The transaction manager deal with several users who write and read data at the same time that is, concurrently.
- Data should be protected from unauthorized access and accidental loss. Inventory control, airline reservations, order fulfilment are the classic examples of the transaction manager.
- Steps involved in the transaction manager are :
  1. Map the class model to the database structures.
  2. Determine the units of concurrency.
  3. Determine the unit of transaction.
  4. Design concurrency control for transactions.

5.2.9 Designing Algorithms

- As far as functional model is concerned, an algorithm should be defined and designed for each of the function or operation involved within a proposed software system.
- The software system analysis specification refers to the exact functionality or operation involved within a particular function while an algorithm tells us how that functionality or operation can be achieved.

- In order to define an algorithm, **following steps should be followed**:
  1. Select an algorithm in such a way that, it reduces the implementation costs of a particular function or operation.
  2. Suitable data structures should be selected and used for defining and implementing an algorithm.
  3. New internal classes and operations should be defined and designed if it is essential.
  4. Assign operations to suitable classes involved within a software system scenario.
- So, initially it is quite essential and is the best practice to define and design detailed algorithms for each and every functionality and operation of a proposed software system.

5.2.10 Design Optimization

- After designing an algorithm, next step is to implement an algorithm and optimize it. The system analysis archetype is used as the basic framework for implementation of a proposed software system.
- The system design model supports system analysis model in order to formulate the logical information about the proposed software system.
- Design optimization plays a vital role in the software system modelling by means of design of a correct logic which is then followed by implementation and execution of a particular software system.
- Every module or part of a software system involved within a scenario is quite important for implementation and proper execution of a particular functionality or operation involved within a software system. Basically, the system analysis model provides a base for the system design model.
- The system analysis model refers to the logic of a particular software system whereas the system design model is dedicated and meant for actual implementation and execution of a proposed software system.
- Following activities are involved in the process of design optimization and these activities should be attained in order to accomplish design optimization :
  1. Effective access paths should be made available.
  2. In order to achieve superior efficiency, the computation should be reorganized.
  3. In order to avoid recomputation, intermediary results obtained from the software system should be maintained.

Key Concepts

• System Analysis	• Static object modeling
• Dynamic object modeling	• Software system analyst
• Noun/verb analysis	• Class, Responsibilities & Collaborations (CRC)
• CRC analysis	• Rational Unified Process (RUP)
• Entity class	• Control class
• Boundary class	• Data dictionary
• Association	• Inheritance

• Multiple inheritance	• Mix-in inheritance
• Abstraction	• Package
• Import dependency	• Access dependency
• Use dependency	• Merge dependency
• Trace dependency	• Actor
• Use case	• Initial event
• Final event	• Scenario
• External event	• Design Optimization

Summary

- The static and dynamic archetypes of the proposed software system are designed in the system analysis phase.
- The main motive behind ***system analysis*** is to find out real-world objects and entities in the proposed system scenario and then plotting these objects as the software objects in the system scenario.
- The static object modeling in object oriented analysis comprises of object modeling and class modeling.
- The dynamic object modeling in object oriented analysis consists of state modeling, activity modeling, use case modeling and interaction modeling.
- A class can be defined as a detailed explanation of a group of objects that share the equivalent attributes, relationships and operations.
- A class of a particular object is a fundamental feature of that object.
- For each object, it is possible to identify and distinguish its own class.
- For finding the classes, no stepwise procedure or algorithm is available in the literature.
- So, analysis of classes is the whole sole responsibility of the personality known as ***software system analyst***. The experienced software system analysts may help out for defining the classes of the proposed software system in right manner.
- In the procedure of ***noun/verb analysis*** for finding out the classes, noun phrases or nouns are dedicated for defining classes themselves or for defining attributes of respective classes and verb phrases or verbs depicts operations or responsibilities of the respective classes.
- CRC stands for Class, Responsibilities and Collaborations.
- CRC analysis strategy should be used along with noun/verb analysis process.
- CRC analysis is a two-step procedure: Collecting information and Analyzing information.
- RUP stands for Rational Unified Process.
- Concept of ***analysis class*** is considered in ***RUP***.
- ***Analysis classes*** are the classes that signify an abstraction in the proposed system domain and are mapped to the real-world entities.

- Entity class, control class and boundary class are the types of analysis classes.
- ***Entity class*** gives determined information about the particular entity involved within the system scenario.
- ***Control class*** summarizes use case modeling and helps in describing use case model in brief.
- ***Boundary class*** is used as a facilitator between proposed system and outside environment. It provides a platform for communication and coordination between the system and its environment.
- ***Data dictionary*** is the phrase related to 'data' and is the thing important for modeling all of the components involved in the system scenario.
- The data dictionary specifies attributes, responsibilities, operations and associations.
- ***Association*** is a structural relationship amongst two classes and is static in nature. It shows fixed relationships between classes involved in a system.
- ***Associations*** can be mentioned in terms of ***verb phrases*** or ***verbs***.
- ***Inheritance*** is the mechanism with the help of which specific elements can obtain their behavior and structure from general elements involved within the system scenario.
- For organizing and simplifying classes with the help of inheritance, system analyst should first of all look after responsibilities, operations, and attributes that are common to two or more classes in a group of classes.
- It is possible for a class to have more than one superclass in Unified Modeling Language. That means, a child can have more than one parent and this property is known as ***Multiple Inheritance***.
- ***Multiple inheritance*** is not supported by all of the object oriented languages. C# and Java permits only single inheritance.
- A child class can inherit properties of its superclasses in multiple inheritance.
- Normally, superclasses should not have a parent class in common in order to avoid cycles in the inheritance hierarchy.
- All the superclasses involved in the scenario should be semantically disjoint.
- Multiple inheritance permits mix-in inheritance.
- The main objective of an ***abstraction*** is to segregate those facets of a particular problem which are essential for some purpose and destroy those features which are irrelevant.
- ***Abstraction*** should be purpose specific, for the reason that the purpose defines what is important, and what is not important.
- An abstraction signifies the crucial features of an object that differentiate it from all other types of objects.
- The ***package*** is a widespread mechanism dedicated for organizing all of the building blocks of UML into groups.
- Normally, packages are used for presenting behavioral and structural views of the software system.
- The ***<<import>> dependency*** combine client and supplier namespaces and it implements a public merge.

- The ***<<access>> dependency*** also combine client and supplier namespaces. Only the difference is that, it implements a private merge.
- The ***<<use>> dependency*** represents the interrelationship between the elements within the packages and not the packages themselves.
- The ***<<merge>> dependency*** is used in metamodeling only. We should not take into consider this kind of dependency in case of object oriented modeling and design.
- The ***<<trace>> dependency*** typically shows the interrelationship between models (archetypes) instead of depicting the relationship between elements of a system.
- For finding actors of a particular system, following questions should be taken into consideration :
  - Who are the end users of the system?
  - Who are the installers of the system?
  - Who provides information to the system?
  - Is there any other cooperating or interacting system available in the scenario?
  - Who maintains the system?
- From the business point of view, each actor should be named uniquely.
- Actors are always external to the system.
- In UML, the system requirements and functionality of the system are depicted with the help of use cases.
- ***A use case*** is defined as a set of scenarios that collectively work to achieve a common user goal. It outlines a sequence of interactions amongst one or more actors and the system itself.
- Use cases are meant for specification of the interaction between the system itself and end users of the system which are termed as actors in UML.
- Use case offers a detailed description of how the system is used.
- More than one actors can be a part of the complex use cases.
- Things that occur at a particular instance of time are known as ***events***.
- An event owns an activity or operation along with its time and location.
- Defining ***initial events*** is simply nothing but the demand of service from respective use cases during execution.
- The ***final event*** simply determines the boundary line for a particular activity or set of activities in some sequence. i.e.; it tells us where to stop or where to terminate.
- A ***scenario*** is nothing but the ordered set of events or activities that occurs in between objects and it illustrates behavior.
- Scenario can be defined as a definite path through a use case.
- ***Activity diagram*** helps us for representing step by step flow of execution of the system by considering the system as a whole.
- An activity diagram can be used to provide a detailed description of the use case.
- Normally, a high level activity node is used to show a single use case and then it can be separated into a separate activity diagram.

- Use cases in use case diagram are replaced by activity nodes in the activity diagram, relations like association, generalization, include and extend in use case diagram are replaced by loops, arcs and decision points depending on the situation in the system scenario. An activity node depicting a use case can also be used for showing a link to an inclusion use case or exclusion use case.
- We can decompose a system into subsystem by merging partitions and layers.
- Partitions can be layered and layers can be partitioned.
- All objects are concurrent or synchronized in the system analysis model, as in the real world and in hardware.
- As far as implementation is concerned, all of the software objects are not concurrent, since one processor can support many objects.
- Two objects are inherently concurrent if they can receive the event at the same time deprived of interacting.
- A ***thread of control*** is a path through a set of state diagrams on which only a single object is active at a time.
- A thread remains within a state diagram unless and until an object sends an event to another object and waits for another event.
- The thread passes to the receiver of the event until it ultimately returns to the original object.
- The thread splits if the object sends an event and continues executing.
- On each thread of control, only single object is active at a time.
- We can implement thread of control as a particular task in computer system.
- The number of essential processors required totally depends on the speed of the machine and the volume of calculations.
- Numbers of substitutes are available for managing the data storage that we can use distinctly or in combination. For instance, databases, files, data structures, etc.
- Different types of data stores offer trade-offs between cost, access time, capacity, cost, reliability and access time.
- ***Hardware control*** closely matches the analysis model however there are quite a lot of ways for implementing and executing control in a software system.
- ***Internal control*** concerns the flow of control contained by a process.
- In a ***procedure-driven sequential system***, control exists within the program code.
- Procedure request external input and then wait for it; when input arrives, control resumes within the procedure that made the call.
- The location of the program counter and the stack of procedure calls and local variables define the system state.
- The main benefit of procedure-driven control is that, it is quite easy to implement with conventional languages while the drawback is that it necessitates the concurrency inherent in objects to be mapped into a sequential flow control.
- In an ***event-driven sequential system***, control exists within a dispatcher or monitor that the language, subsystem, or operating system offers.

- In a ***concurrent system***, control exist simultaneously in several independent objects, each a separate task.
- Internal object interactions are similar to external object interactions since we can make use of the same implementation mechanisms.
- The software system designer should set priorities that will be used to guide trade-offs for the rest of the software system design.
- Number of architectural styles are commonly used in existing software systems :
  - Batch Transformation
  - Continuous Transformation
  - Interactive Interface
  - Dynamic Simulation
  - Real Time System
  - Transaction Manager
- In ***batch transformation***, the information transformation is executed once on a complete input set.
- A ***continuous transformation*** is a system in which the output of the system is aggressively dependent on varying inputs. A continuous transformation updates outputs frequently.
- An ***interactive interface*** is a system that is conquered by interactions amongst the external agents and the system itself. The external agents can be devices or humans.
- ***Dynamic simulation*** tracks or models real world objects.
- The ***real time system*** is an interactive system with close-fitting or tight time constraints on actions.
- A ***transaction manager*** is a system dedicated for retrieval and storage of data.

Chapter Ends...

❑❑❑