Use case methods – Object Analysis – Object relations Attributes – Methods – Class and Object responsibilities – case studies

Object Oriented Analysis: -
- The main **objective** of analysis is to capture a complete, unambiguous and consistent picture of requirements of system and what system must do to satisfy users’ requirements and needs
- **Analysis** is the process of transforming a problem definition from a fuzzy set of facts and myths into a coherent statement of system’s requirements
- **Why Analysis is difficult activity?** Analysis is a creative activity that involves understanding the problem, its associated constraints and methods of overcoming those constraints. This is an iterative process that goes on until problem is well understood
  - Norman explains three most common sources of requirement difficulties: Fuzzy descriptions, Incomplete requirements and unnecessary features
  - A common problem that leads to requirement ambiguity is fuzzy and ambiguous description such as “fast response time” or “very easy and very secure updating mechanisms”.
  - Incomplete requirements represent certain necessary requirements are not included for variety of reasons including cost, politics, etc.. Due to iterative nature of OOA & unified approach, most of incomplete requirements can be identified in subsequent tries
  - In addressing features, every additional feature could affect performance, complexity, stability, maintenance and support costs of an application. A number of other factors can affect design of an application. Analysis is difficult activity until understanding problem domain and implementation in s/w. Experience always the best teacher

**Use Case driven OOA: Unified Approach:** -
- OOA phase of UA uses actors & use cases to describe system from users’ perspective. Actors are external factors that interact with system; use cases are scenarios – describe how actors use system
- The OOA process consists of following steps: -
  1. **Identify the actors:** * who is using system? * In new case, who will be using the system?
  2. Develop a **simple business process model** using UML activity diagram
  3. **Develop the use case**:
     - What are the users doing with the system?
     - In case of new system, what will users be doing with the system?
     - Use cases provide us with comprehensive documentation of system under study
  4. **Prepare interaction diagrams:** * Determine the sequence, * Develop collaboration diagrams
  5. **Classification** – develop a static UML class diagram: * Identify classes, * Identify relationships
     * Identify attributes * Identify methods
  6. **Iterate and refine:** if needed, repeat the preceding steps

**Use – Case Model:** -
- A use case is an **interaction** between users and a system; it captures the goal of users and responsibility of system to its users. The use – case model describes the uses of system and shows the courses of events that can be performed. It expresses what the business or application will do and not how; that is responsibility of UML class diagram
- **Under the microscope:** A use case is a sequence of transactions in a system whose task is to yield results of measurable value to an individual actor of system. Key words of this classification
  - **Use–Case:** It is special flow of events through the system. It is suggested that, to make a use–case model meaningful, we must group courses of events and call each group a use–case class
  - **Actors:** An actor is user playing a role with respect to system. It is important to think about roles rather than just people & their job titles. An actor is an external system that needs some information from current system. They get value from use case or just participate in use case
  - **In a system:** This simply means that the actors communicate with the system’s use case
  - **A measurable value:** A use case must help the actor to perform a task that has some identifiable value. Eg: Performance of use case in terms of price or cost
  - **Transaction:** It is an atomic set of activities that are performed either fully or not at all. It is triggered by stimulus from an actor to system of by a point in time being reached in the system
- **Uses & Extends Associations:** The extends association is used when we have one use case that is similar to another use case but does bit more or is more specialized; in essence it is like a subclass.
- To avoid describing a subflow more than once in several use cases, we can extract common subflow and make it a use case of its own. The relationship among other use cases and this new extracted use case is called uses association. An abstract use case is not complete and has no initiation actors but is used by a concrete use case, which does interact with actors
- **Identifying Actors:** The term actor represents the role a user plays with respect to system. When dealing with actors, it is important to think about roles rather than people or job titles
Guidelines for finding Use Cases:  The steps for finding use cases are
1. For each actor, find tasks & functions that actor should be able to perform or that system needs the actor to perform. Use case should represent a course of events that leads to a clear goal
2. Name the use cases
3. Describe use cases briefly by applying terms with which user are familiar.  This makes description less ambiguous

Developing an Effective Documentation: - 
1. The key in developing effective documentation is to eliminate all repetition; present summaries, reviews, organization chapters in less than three pages and make chapter headings task oriented so that table of contents also could serve as an index
2. The documentation depends on organization’s rules & regulations. Most organizations have established standards or conventions for developing documentation. Too little documentation invites disaster; too much documentation, as Blum put it, transfers energy from problem solving tasks to a mechanical and unrewarding activity

Guidelines for developing Effective Documentation: Bell and Evans provide the following guidelines for making documents fit the needs and expectations of our audience
1. Common cover: All documents should share common cover sheet that identifies document, current version, individual responsible for content. For each phase in lifecycle, separate cover encountered.
2. 80 – 20 rule: 80% of work can be done with 20% of documentation. Make sure that 20% is easily accessible & rest 80% is available to those few who need to know
3. Familiar Vocabulary: The main objective here is to communicate with readers & not impress them with buzz words by using common words & comfortable for readers
4. Make document as short as possible: The key in developing an effective manual is to eliminate all repetition; present summaries, reviews, organization chapters in less than 3 pages and make chapter headings task orientated so that table of contents also could serve as an index
5. Organize the document: Use rules of good organization within each section. Most CASE tools provide documentation capability by providing customizable reports.

The purpose of these guidelines is to assist in creating an effective documentation

Object Analysis: -
1. OOA is process by which we identify classes that play role in achieving system goals & requirements. Classification is the process of checking to see if an object belongs to a category or a class which guides us in making decisions about modularization
2. Four alternate approaches for identifying classes & their behaviors in problem domain are the noun phrase approach; the common class patterns; the use-case driven, sequence/collaboration modeling approach; and the Classes, Responsibilities and Collaboration (CRC) approach
3. Noun Phrase Approach: Here, we read through the requirements or use cases looking for noun phrases. Nouns in textual description are considered to be classes & verbs to be methods of classes. As a whole, classes are grouped in to three categories: Relevant classes, Fuzzy classes and Irrelevant classes. The series of steps in this approach are as follows:
   ⇒ Identifying Tentative Classes: Following are guidelines for selecting classes in an application
      Look for nouns, and noun phrases in use cases
      Some classes are implicit or taken from general knowledge
      All classes must make sense in application domain
      Avoid computer implementation classes – defer them to the design stage
      Carefully choose and define class names
   ⇒ Selecting Classes form Relevant & Fuzzy Categories: Following guidelines help in selecting candidate classes from relevant & fuzzy categories of classes in problem domain
      Redundant classes: If more than one word is being used to describe same idea, select one that is most meaningful in the context of system. This part of building a common vocabulary for the system as a whole
      Adjectives classes: Adjectives can be in many ways. An adjective can suggest a different kind of object, different use of the same object or it could be utterly irrelevant
      Attribute classes: Tentative objects that are used only as values should be defined or restated as attributes and not as a class
      Irrelevant classes: Each class must have a purpose and every class should be clearly defined and necessary. Classes which cannot be given statement of purpose are eliminated
   ⇒ Elimination & Refining: The process of eliminating redundant classes & refining remaining classes is not sequential. It can be done forth & back among steps as of wish
Review redundant classes
Review adjectives

Review irrelevant classes
Review attributes

Review the possible attributes and class purpose in an iterative manner

1. **Common Class Patterns Approach**: It is based on a knowledge base of common classes that have been proposed by various researchers. They have compiled & listed the following patterns for finding the candidate class & object
   - **Concept class**: It encompasses principles that are not tangible but used to organize or keep track of business activities or communications. Eg: Performance is an concept class object
   - **Events class**: These are points in time that must be recorded. Associated with things remembered are attributes such as who, what, when, where, how or why. Eg: Landing
   - **Organization class**: It is collection of people, resources, facilities or groups to which users belong; their capabilities have a defined mission, whose existence is independent of individuals
   - **People class**: It represent different roles users play in interacting with application. It is also known as person, roles and roles played class divided into 2 – users & non-users information
   - **Places class**: Places are physical locations that system must keep information about. Eg: Stores
   - **Tangible things & devices class**: This class includes physical objects or groups of objects that are tangible & devices with which application interacts. Eg: cars, pressure sensors

2. **Use-case driven Approach**: It is considered as problem–driven approach to OOA. Modeling in this approach is recommended aid in finding the objects of a system and is the technique used by UA. The implementation of scenarios recommends UML usage with different instances

3. **CRC Approach**: The Classes, Responsibilities and Collaborators process consists of three steps
   - 1. Identify classes’ responsibilities (and identify classes)
   - 2. Assign responsibilities
   - 3. Identify collaborators
   Classes are identified & grouped by common attributes, which also provides candidates for super classes. Responsibilities are distributed; they should be as general as possible & placed as high as possible in inheritance hierarchy. The idea in locating collaborators is to identify how classes interact. Classes (cards) that have a close collaboration are grouped together physically

4. **Naming Classes**: It is an important activity. The class should describe a single object, so it should be a singular noun or an adjective and a noun. A general rule for naming classes is that you use names with which users or clients are comfortable. Choose class names from standard vocabulary for subject matter

**Object relationship Attributes**: –
- In an OO environment, objects take on an active role in a system. The relationship among objects is based on assumptions each makes about other objects, including what operations can be performed and what behavior results
- In general, three types of relationships among objects exist such as Association; Super–sub structure (also known as generalization hierarchy); Aggregation and a–part–of structure

**Associations**:
- It represents a physical or conceptual connection between two or more objects. This information will guide us in designing classes. Answers to following questions help us identify association.
  The needed in approach is flexibility.
  - Is class capable of fulfilling the required task by itself?
  - If not, what does it need?
  - From what other class can it acquire what it needs?

**Guidelines for identifying Association**
- A dependency between 2 or more classes may be an association. It is often corresponds to a verb or prepositional phrase, such as part of, next to, works for, or contained in
- A reference from one class to another is an association. Some associations are implicit or taken from general knowledge

**Common Associative Patterns**:
- * Location association – next to, part of, contained in.
- * Communication association – talk to, order to

**Eliminate** Unnecessary Associations by Implementation association, Ternary associations, Directed actions (or derived) association. Choose association names carefully. Add role names where
appropriate, especially to distinguish multiple associations. These often are discovered by testing access paths to objects

**Super – sub class relationships:**
- This relationship represents the *inheritance relationships* between classes. Class inheritance is useful for a number of reasons. These relationships are also called as *generalization hierarchy*; allow objects to be built from other objects. Such relationships allow us to explicitly take advantage of commonality of objects when constructing new classes.
- The following *guidelines* are used in identifying this sort of relationships in application
  - **Top-down:** Avoid excessive refinement; Specialize only sub classes have significant behavior
  - **Bottom-up:** Look for classes with //lr attributes or methods & group them to an abstract class
  - **Reusability:** Move attributes and behaviors (methods) as high as possible in hierarchy. The balancing act can be achieved through several iterations. This process ensures objects reuse
  - **Multiple Inheritance:** Avoid excessive use of multiple inheritance which brings with it complications such as how to determine which behavior to get from which class. It is also difficult to understand programs written in a multiple inheritance system

**A–Part–Of Relationships – Aggregation:**
- It represents the situation where a class consists of several component classes. A class that is composed of other classes behaves differently. Two major properties are
  - **Transitivity:** The property where, if A is part of B and B is part of C, then A is part of C
  - **Anti-symmetry:** The property where if A is part of B, then B is not part of A
- A clear distinction between part & whole can help us determine where responsibilities for certain behavior must reside. In UML, a filled diamond signifies the strong form of aggregation which is composition. *Eg.:* Collection as hollow diamonds & solid diamond for strong composition.
- To identify a–part–of structures, Coad & Yourdon provide following guidelines:
  - **Assembly:** It is constructed from its parts and & an assembly–part situation physically exists
  - **Container:** A physical whole encompasses but is not constructed from physical parts
  - **Collection–member:** A conceptual whole encompasses parts that may be physical or conceptual

**Class Responsibility:**
- Identifying Attributes & Methods is like finding classes, still difficult activity an iterative process. Once again use cases & other UML diagrams will be our guide for identifying attributes, methods & relationships among classes
- In short, *responsibilities* identify problems to be solved. Responsibilities are meant to convey a sense of purpose of an object & its place in system. The responsibilities of an object are all the services it provides for all the contracts it supports
- *Attributes* are things an object must remember such as color, cost & manufacturer. Identifying attributes of a system’s class starts with understanding the system’s responsibilities
- Guidelines for defining Attributes of classes in use cases:
  - Attributes usually correspond to nouns followed by prepositional phrases such as cost of soup. Attributes also may correspond to adjectives or adverbs
  - Keep the class simple; state only enough attributes to define the object state
  - Attributes are less likely to be fully described in problem statement. You must draw on our knowledge of application domain & real world to find them
  - Omit derived attributes. *Eg.:* Do not use time elapsed since order
  - Don’t carry discovery of attributes to excess. More attributes can add in subsequent iterations

**Object responsibilities:**
- In an OO environment, every piece of data or object is surrounded by a rich set of routines called *methods*. These methods do everything from printing the object to initializing its variables
- Every class is responsible for storing certain information from the domain knowledge. If an object requires certain information to perform some operation for which it is responsible, it is logical to assign it the responsibility for maintaining the information
- In *UML sequence diagram*, objects involved are drawn as vertical dashed lines. An event is considered to be an action that transmits information. In other words. These actions are operations that objects must perform & as in attributes, methods also can be derived from scenario testing
- In additional, operations (*methods or behavior*) in OO system usually correspond to queries about attributes & associations of objects. Therefore, methods are responsible for managing the value of attributes such as query, updating, reading & writing
Case studies: -

- The ViaNet Bank ATM system: Identifying classes by using Noun Phrase Approach. Here we start by reading use cases & applying the principles discussed for identifying classes.

- Initial list of Noun Phrases: Candidate classes are identified from initial study of use cases of bank.

<table>
<thead>
<tr>
<th>Account</th>
<th>Account Balance</th>
<th>Amount</th>
<th>Approval process</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM card</td>
<td>ATM machine</td>
<td>Bank</td>
<td>Bank Client</td>
</tr>
<tr>
<td>Card</td>
<td>Cash</td>
<td>Check</td>
<td>Checking</td>
</tr>
<tr>
<td>Checking a/c</td>
<td>Client</td>
<td>Client’s a/c</td>
<td>Currency</td>
</tr>
<tr>
<td>Dollar</td>
<td>Envelope</td>
<td>Four digits</td>
<td>Fund</td>
</tr>
<tr>
<td>Invalid PIN</td>
<td>Message</td>
<td>Money</td>
<td>Password</td>
</tr>
<tr>
<td>PIN</td>
<td>PIN code</td>
<td>Record</td>
<td>Savings</td>
</tr>
<tr>
<td>Savings a/c</td>
<td>Step</td>
<td>System</td>
<td>Transaction</td>
</tr>
<tr>
<td>Transaction</td>
<td>History</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Transaction History**

- The following irrelevant classes eliminated because they do not belong to problem statement: Envelope, Four Digits and Steps – Strikeout the eliminated classes (3) in above candidate classes.

- Reviewing redundant classes & building a common vocabulary: The following are different class names that are being used to refer to same concept, one class of group is chosen and rest eliminated:

  - Client, Bank Client = Bank Client
  - Account, Client’s a/c = Account
  - PIN, PIN Code = PIN
  - Checking, Checking a/c = Checking a/c

  4 classes are been eliminated in this step (Client, PIN code, Client’s a/c, Checking).

- Reviewing classes containing Adjectives: We have no classes containing Adjectives to eliminate.

- Reviewing possible attributes: Next review focuses on identifying the noun phrases that are attributes, not classes. The noun phrases used only as values should be restated as attributes. The following attributes are been identified and eliminated from the list of classes (6 classes):

  - **Amount**: A value, not a class
  - **Account Balance**: An attribute of a/c class
  - **Invalid PIN**: A value, not a class
  - **Password**: An attribute of Bank Client class
  - **PIN**: An attribute of Bank Client class
  - **Transaction History**: An attribute of Transaction class

- Reviewing Class Purpose: The classes that add no purpose to system have been deleted form list. The candidate classes are these:

  - **ATM Machine class**: Provides an interface to Via Net bank
  - **ATM Card class**: Provides a client with a key to a/c
  - **Bank Client class**: Client is an individual that has checking a/c & possibly a saving a/c
  - **Bank class**: It is a repository of accounts & processes the a/c’s transactions
  - **Account class**: An account is a formal class; it defines the common behaviors that can be inherited by more specific classes such as Checking a/c & saving a/c
  - **Checking Account class**: It models a client’s checking a/c & provides more specialized withdrawal service
  - **Saving Account class**: It models a client’s savings a/c
  - **Transaction class**: Keeps track of transaction, time, date, type, amount, balance

- Thus, at last right from 33 initially identified classes, only 8 classes are cut listed by this approach.

Questions: -

- **5 marks**
  1. Why is documentation an important part of analysis?
  2. What is the point should be considered while naming a class?
  3. What is use case modeling? Explain.
  4. Describe the difference between a method and a process.
  5. Write short notes on (i) Generalization, (ii) Association.

- **10 marks**
  1. Discuss use case driven object – oriented analysis in a unified approach in detail.
  2. Discuss CRC approach for identifying classes in a problem domain.
  3. Explain the method of preparing documentation in detail.
  4. Discuss the noun phrase approach for identifying classes
Unit – I → Questions : -

5 Marks

1. What is pattern? Explain
2. What is static binding? What is dynamic binding? Explain.
3. How is software verification different from validation?
4. How does object-oriented software development promote reusability?
5. What are the phases of OMT? Explain.
6. What is UML? What is the importance of UML?
7. What are the advantages of object-oriented development?
8. What is use case modeling? Explain.

10 Marks

1. Discuss component based development for a software development process?
2. Explain layered approach to software development in detail
3. What is modeling? Why is it necessary? What are the advantages of modeling?
4. Discuss use case diagram with an example.
5. Discuss use case driven object-oriented analysis in a unified approach in detail.
6. Explain object oriented system development a use-case approach in detail.
7. Discuss unified approach of s/w development.
8. Explain UML class diagram and its various related notations.
9. Discuss the necessity for modeling and its advantages.

Other Specified Questions: -

5 – Marks

1. Discuss about metaclass in detail.
2. What are the advantages of object-oriented development?
3. What are the process of unified approach? Explain.
4. What is meta-model? Is understanding a meta-model important?