UNIT I- Introduction: Role of Analysis and Design in Software Development – Meaning of Object Orientation - Overview of Various OOAD Methodologies - Goals of UML. Use case Modeling: Actors and Use Cases - Use Case Relationships - Writing Use Cases formally - Choosing the System Boundary - Finding Actors - Finding Use Cases - Use of Use Cases for Validation and Verification - Use Case Realization.

1 INTRODUCTION

1.1 Role of Analysis and Design in Software development

Software development includes taskslike:

- Defining theproblem,
- Determining scope and vision,
- Gathering detailedrequirements,
- Designing,
- Programming thesystem
- Testing the developedsystem,
- Delivering the productand
- Even maintainingit.
- In SSAD approach, requirements- gathering isfollowed by analyzing the requirements.
- End-product of requirement analysis can be a Data-Flowdiagram.

Analysis

- Investigateproblem
- Scrutinizerequirements
- Emphasis on "What is supposed to be done" rather than "how to doit".
- If System to be developed is "Video Library Management System"then
- What is the software developer expected to do for the client? If manual system present, then study it and get an overview of what he is expected todo.
- Using various tools and techniques, elicit requirements from thecustomer.
- Make customer aware of the additional capabilities of computer softwaretechnology.

Design

- Provide a blueprint for asolution.
- Move from "what is to be done" to "how it is to bedone".
- Two types of designs: high-level and low-level design.

- In high-level design, structure of the solution, interfaces, modules, input/output formodules

- In detailed design, internal logic of themodules,The data structures to be used etc.

1.2 Meaning of Object Orientation?

- A term used to describe the object-oriented method of buildingsoftware.
- Data treated as the most importantelement.
- It cannot flow freely around thesystem.
- Restrictions placed on the number of units that can manipulate thedata.
- Encapsulation
- Abstraction, ImplementationHiding
- Inheritance, Dynamic binding,Polymorphism
- Overriding andOverloading

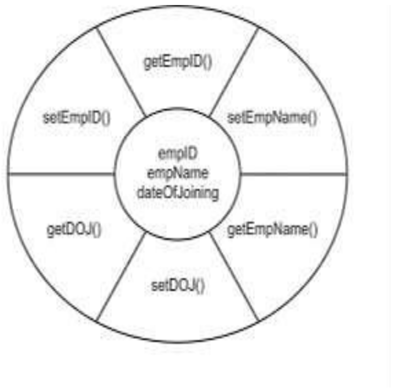
Encapsulation (SSAD)

- The problem: Write a program to find the Permutation and combination by taking n and r from theuser.
- nPr=n!/(n-r)! andnCr=n!/((n-r)!*r!).
- Factorial computation is necessary (more than once). So, in a procedural programming language like C, write the instructions to compute the factorial within a function **only once.**
- Invoke it multiple times by passing different actual arguments. Thus a function is used as a unit.

Encapsulation (OO)

- In OO, a class groups related attributes and operations together. The outside world interacts with the data stored in the variables that represent the attributes of the class only through the operations of thatclass.

Encapsulation



Abstraction

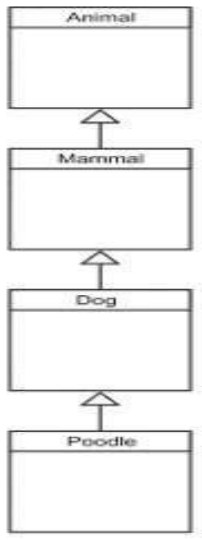
- Consider aradio.

- Radio is oneunit.
- User not intimidated by complex subunits (StationManagement,Power-Management, Volume Control) and their working.
- To him, the radio works as one abstract unit (with externalbuttonsto represent these sub-units).

Abstraction

- Similarly, the **class** is one abstractunit.
- To perform a task that involves an object of that class, send a message to the object asking it to execute the respectiveoperation.
- Implementation Hiding
- Manner in which data is stored in an object is hidden from the outsideworld.
- Suppose dateOfJoining attribute value has to be stored in objecte1.
- Either as dd/mm/yy or mm/dd/yy or in some othermanner.
- To retrieve date of joining, the user is only interested to get the value in a format understandable tohim.
- Inheritance
- Property by which one class inherits the features of anotherclass.
- E.g. Poodle is a Dog, A Dog is a Mammal and a Mammal is ananimal.
- A Poodle also has highly specialized features, which distinguishes it from other dogs say Dobermans andAlsatians.

Inheritance



Dynamic Binding

- Dynamic/runtime/late binding is a technique in which the piece of code to be executed is determined only atrun-time.
- Quadrilateral Q = new Quadrilateral();
- Rectangle R = newRectangle();
- Square S = newSquare();
- If users clicks on Rthen
- Q =R;
- Else
- Q =S;
- Q.getArea();
- No knowledge at compile time which piece of code to execute.
- Depends onuser.

Polymorphism

- Ability to take more than oneform.
- Consider a class Quadrilateral with calculateArea().
- Classes Square and Rectangle inherit from Quadrilateral.
- Area of Square= length*length

- Area of Rectangle= length *breadth
- Thus,Operation with same name takes two different forms in classes Squareand
- Rectangle.

Overriding

- **Overriding** of a method in a class is the redefinition of a method in the sub-classes of thatclass.
- Suppose there was a complex definition of the method calculateArea() in class Quadrilateral. Let class Rectangle inherit from classQuadrilateral.
- Let the inherited operation calculateArea() be redefined in class Rectangle as length * breadth. Thus calculateArea() is overridden within the classRectangle.
- If an object of class Rectangle is created and operation calculateArea() is invoked, the Rectangle's calculateArea() method will beexecuted

Overloading

- Term used when several methods **within the sameclass**have the same name but different signatures.
- Signature means the number and/or type ofinputparameters in the method.
- Consider the following operation named sum in classAdd.
  class Add {
  int sum(inta,int b);
  double sum(double a,double b);
  float sum(float a,float b);
  int sum(inta,intb,int c)
  }
- Return type is notconsidered.

What is Oriented Analysis and Design?

- In OO Analysis, in addition to understanding the requirements, Identify the concepts in the domain. For example, in the VLMS, Patron, DVD, Cassette etc are the concepts(objects).
- In OO Design, Find software objects. Now a conceptual object is a DVD. Its attributes can be capacity, the title of the film that is stored on it, film certificationetc.

1.3 Overview of Various Methodologies

- Booch
- Coad and Yourdon
- Fusion
- Jacobson: Objectory andOOSE
- LBMSSEOO
- Rumbaugh OMT
- Shlaer and Mellor OOAnalysis

Booch Methodology

- Supported by various tools like Visio and RationalRose.
- Covers requirement and domain analysis and its major strength is indesign.

Coad and Yourdon Methodology

- It focuses on analyzing the business problems. The analysis process has five stages:
    - Subjects
    - Objects
    - Structures
    - Attributes
    - Services
- In design, these five activities aresupplanted by and refined into four components:
    - Problem Domain Component
    - Human Interaction Component
    - Task Management Component
    - Data ManagementComponent

Fusion

- Developed by Derek Coleman andothers.
- Borrowed and adapted ideas from other methodologies.
- Incorporated some major ideas from Booch, Jacobson, Rumbaugh, and others, and explicitly rejected many other ideas from these methodologies.
- Fusion's pragmatic approach holds considerable potential for client/serverapplications.

Jacobson: Objectory and OOSE

- Objectory isproprietary.
- Object-Oriented Software Engineering (OOSE) is a simplified version of Objectory.
- Object modeling and many other OO concepts in Objectory and OOSE similar to OO concepts in othermethodologies.

- Major distinguishing feature is the use case.
- Drawback with OOSE: assumption that all sequences can be expressedinusecases.

LBMS SEOO

- Systems Engineering OO (SEOO) is a proprietary methodology andtoolkit.
- Tightly integrated with Windows 4GLs such as PowerBuilder.
- Four major components of the SEOO methodology are: Work-breakdown structures and techniques, an object modeling methodology, GUI design techniques and relational databaselinkages.

Rumbaugh OMT

- **Assumes that a requirements specification exists. Analysis consists of building three separatemodels:**
- The Object Model (OM)
- The Dynamic Model(DM)
- The Functional Model(FM)

Shlaer and Mellor OO Analysis

- Information model describing objects, attributes, andrelationships.
- A state model describes the states of objects and the transitions betweenthem
- Data-flow diagram that shows the processmodel.

1.4 GoalsofUML

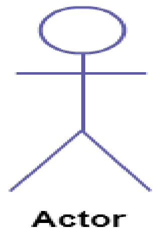
- **Expressive visual modeling language(Visibility)** -Provide users with a ready-to-use, expressive visual modeling language to develop and exchange meaningful models. The UML specifications contain provisions to model software in a variety of ways. Use case diagrams can be drawn after writing the usecases to have a visual feel of the various actors and their requirements from the system. The class diagrams can be drawn to design a solution to the system. The activity diagrams indicate the sequence of activities in the business process and situations where the sequence need not be imposed and activities can occur in any order. The sequence and collaboration diagrams indicate how the objects communicate by passing messages to each other. The state diagram indicates the states of the objects, how the states change in response to the arrival of an event, and what actions need to be carried out as the state changes. The component and deployment diagrams provide for modeling the architectureof the system. All these models help the developer in understanding the system and developing efficient software.
- **Extensibility and Specialization mechanisms(Extensibility) -** Furnish extensibility and specialization mechanisms to extend the core concepts.OMG expects that the UML will be tailored as new needs are discovered and for specific domains. At the same time, we do not want to force the common core concepts to be redefined or re-implemented for each tailored area. Therefore, we believe that the extension mechanisms should support deviations from the common case, rather than being required to implement the core modeling concepts themselves. The core concepts should not be changed more than necessary.
- **Independence from any particular programming language and development process (Independence)** – UML is independent of any particular programming language and development process. The UML can be used to develop models irrespective of the language as well as process followed to develop it. The Unified Process is well suited for UML and forward engineering tools can convert a UML class diagram into Java classes.
- **Encouragement to the growth of OO tool markets(OO tool markets) -** By enabling vendors to support a standard modeling language used by most users and tools, the industry benefits. While vendors still can add value in their tool implementations, enabling interoperability is essential. Interoperability requires that models can be exchanged among users and tools without loss of information. This can only occur if the tools agree on the format and meaning of all the relevant concepts. Using a higher meta-level is no solution unless the mapping to the user-level concepts is included in thestandard.

- **Integration of bestpractices** - A key motivation behind the development of the UML has been to integrate the best practices in the industry, encompassing widely varying views based on levels of abstraction, domains, architectures, life cycle stages, implementation technologies, etc. The UML is indeed such an integration of best practices.

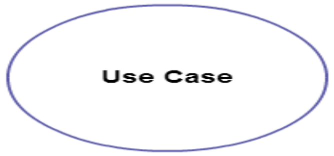
1.5 Use case Modeling: Actors and Use Cases

- Use case modeling is a useful tool for requirements elicitation. It provides a graphical representation of the software system's requirements.
- The key elements in a use case model are **actors** (external entities), and the **use cases** themselves. In outline, a use case is a unit of functionality (a requirement), or a service, in the system. A use case is not a process, or program, or function.
- Because use case models are simple both in concept and appearance, it is relatively easy to discuss the correctness of a use case model with a non-technical person (such as a customer).
- Use case modeling effectively became a practicable analysis technique with the publication of Ivar Jacobson's (1991) book "Object-oriented software engineering: a use case driven approach". Jacobson has continued to promote this approach to system analysis to the present day, and it has now been formalized as part of the UML. However, use case modeling is not very different in its purpose and strategy from earlier techniques, such as structured viewpoint analysis.
- **Use case modeling in the UML specification**
- The Unified Modeling Language (UML) represents a deliberate attempt to standardise the modeling notation used in software engineering, particularly object-oriented development. The widespread uptake of the UML is a result largely of two factors. First, it is driven by some of the most influential proponents of object-oriented development, including James Rumbaugh, Grady Booch, and Ivar Jacobson. Second, it has broad support from major business concerns in the software industry, including Microsoft, IBM, Hewlett-Packard and Oracle.
- The notation specified for use case modeling by the UML is not very different from that originally proposed by Jacobson, the
- UML is under continuous development, and at the time of writing the latest version is 2.2. The definitive reference for the UML notation is the UML specification, which is available from the Object Management Group's Web site. However, while this is an authoritative document about the UML, it is not a good document from which to learn about the UML.
- It is important to understand that the UML is a specification for a modeling language. It is most emphatically not a software design methodology. Although the UML states the symbols that are to be used in use case modeling, and how they are to be interpreted, it does not say when, or even if, use case modeling should be applied.
- **Actors**
- An actor is any entity, human or otherwise, that is external to the system being designed. Two symbols are available in the UML specification:

- An actor can be represented by a stick figure.



- Alternatively, an actor may be represented by a class with the «actor» stereotype.
- **Use Case**
- A use case represents a function or an action within the system. It’s drawn as an oval and named with the function.

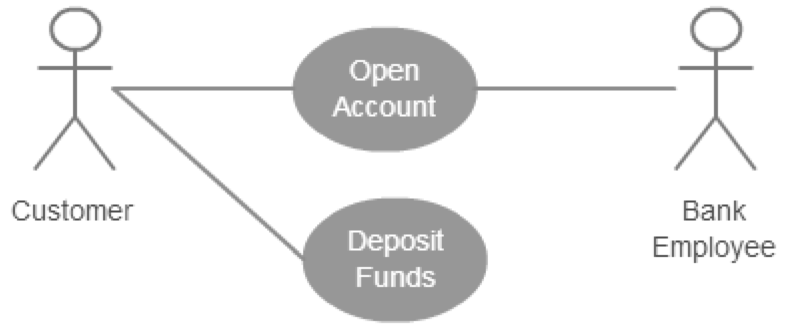


1.6 Use Case Relationships
There can be 5 relationship types in a use case diagram.

- Association between actor and use case
- Generalization of an actor
- Extend between two use cases
- Include between two use cases
- Generalization of a use case

Association between Actor and Use Case:

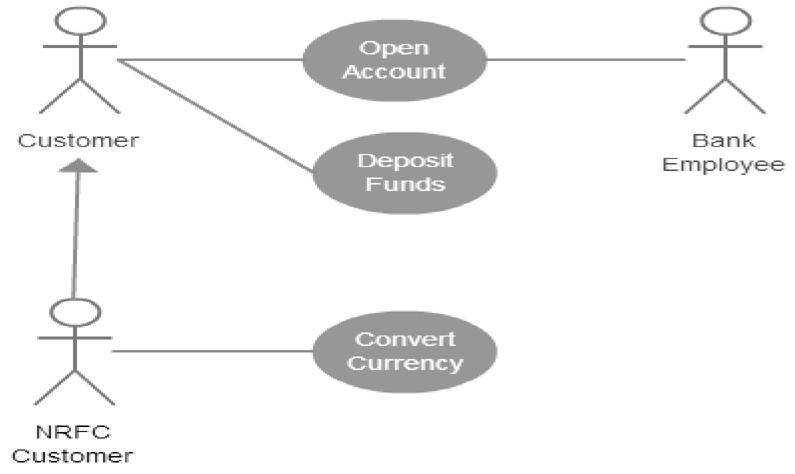
- An actor must be associated with at least one use case.
- An actor can be associated with multiple use cases.
- Multiple actors can be associated with a single use case.



Different ways association relationship appears in use case diagrams

Generalization of an Actor

- Generalization of an actor means that one actor can inherit the role of the other actor. The descendant inherits all the use cases of the ancestor.
- The descendant has one or more use cases that are specific to that role. Let's expand the previous use case diagram to show the generalization of an actor.

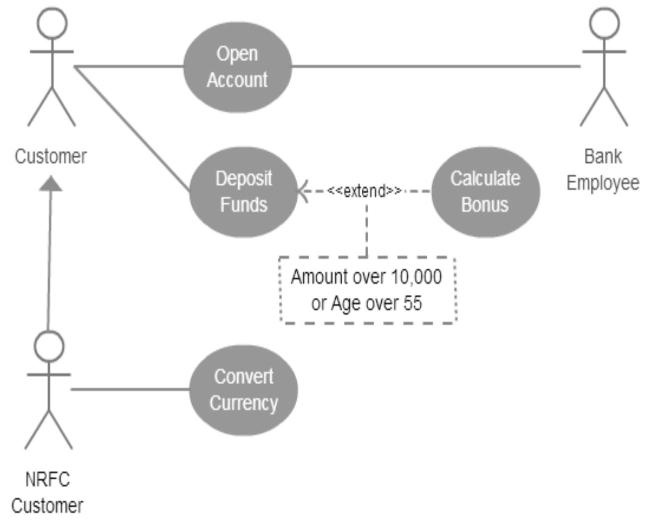


A generalized actor in a use case diagram

Extend Relationship between Two Use Cases

- Many people confuse the extend relationship in use cases. As the name implies it extends the base use case and adds more functionality to the system. The extending use case is dependent on the extended (base) use case. In the below diagram the "Calculate Bonus" use case doesn't make much sense without the "Deposit Funds" use case.

- The extending use case is usually optional and can be triggered conditionally. In the diagram, you can see that the extending use case is triggered only for deposits over 10,000 or when the age is over 55.
- The extended (base) use case must be meaningful on its own. This means it should be independent and must not rely on the behavior of the extending use case.

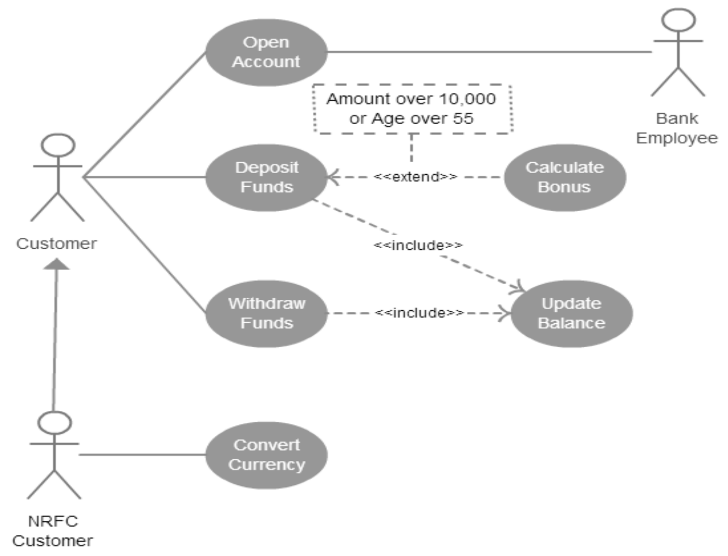


Extend relationship in use case diagrams

- For example, in an accounting system, one use case might be “Add Account Ledger Entry”. This might have extending use cases “Add Tax Ledger Entry” and “Add Payment Ledger Entry”. These are not optional but depend on the account ledger entry. Also, they have their own specific behavior to be modeled as a separate use case.

Include Relationship between Two Use Cases

- Include relationship show that the behavior of the included use case is part of the including (base) use case. The main reason for this is to reuse common actions across multiple use cases. In some situations, this is done to simplify complex behaviors. Few things to consider when using the <<include>> relationship.
- The base use case is incomplete without the included use case.
- The included use case is mandatory and not optional.



Includes is usually used to model common behavior

Generalization of a Use Case

- This is similar to the generalization of an actor. The behavior of the ancestor is inherited by the descendant. This is used when there is common behavior between two use cases and also specialized behavior specific to each use case.
- For example, in the previous banking example, there might be a use case called “Pay Bills”. This can be generalized to “Pay by Credit Card”, “Pay by Bank Balance” etc.

1.7 Writing Use cases formally

A formal template can have the following fields:

1. Primary Actor: He is the principal actor who will initiate the use case.
2. Stakeholders and interests: These are people, and even entities, who are interested in the system.
3. Pre and Post conditions: A pre-condition describes something which must always be true before beginning a scenario in a use case. It is not tested with in the use case. Post condition is a condition that must be true at the successful of the use case. Here success implies success of main as well as alternate flows.
4. Main success scenario: The actor successfully fulfills his goal by interacting with the system.
5. Alternate flow: They are the other scenarios described.
6. Special requirements: Here, non-functional requirements or any constraints relating to the use case may be recorded.
7. Technology and data variations list: In this, any variation in the method to perform a task is recorded.

1.8 How to choose the system boundary

- Choose a System Boundary
- Determine the edges of the system being designed – is it the software, the software and hardware? Does it include the person using the system?
- We are determining the Domain for the system
- In this case, the system under design is the software and hardware associated with it
- The cashier, store databases, payment authorization services, etc. are outside the boundary of the system
- Finding the actors also means that you establish the boundaries of the system, which helps in understanding the purpose and extent of the system. Only those who directly communicate with the system need to be considered as actors. If you are including more roles than that in the system's surroundings, you are attempting to model the business in which the system will be used, not the system itself.

1.9 Finding Actors

- Finding actors is one of the first steps in defining system use. Each type of external phenomenon with which the system must interact is represented by an actor. To find the actors, ask the following questions:
  - Which user groups require help from the system to perform their tasks?
  - Which user groups are needed to execute the system's most obvious main functions?
  - Which user groups are required to perform secondary functions, such as system maintenance and administration?
  - Will the system interact with any external hardware or software system?
- Any individual, group or phenomenon that fits one or more of these categories is a candidate for an actor.
  - To determine whether you have the right (human) actors is sufficient for their needs.

1.10 Finding use cases

Choose the system boundary

- What you are building?
- Who will be using the system?
- What else will be used that you are not building?

Find primary actors and their goals

- Brainstorm the primary actors first
- Who starts and stops the system?
- Who gets notified when there are errors or failures?

Define use cases that satisfy user goals

- Prepare an actor-goal list (and not actor-task list)
- In general, one use case for each user goal
- Name the use case similar to the user goal

The best way to find use cases is to consider what each actor requires of the system.

- What are the primary tasks the actor wants the system to perform?
- Will the actor create, store, change, remove, or read data in the system?
- Will the actor need to inform the system about sudden, external changes?
- Does the actor need to be informed about certain occurrences in the system?
- Will the actor perform a system start-up or shutdown?

1.11 Use of use Cases for Validation and Verification

- Use Cases can be extensively used for testing. Testing involves both validation and verification.
- Validation: – "Are we building the right system?" – Does our problem statement accurately capture the real problem? – Did we account for the needs of all the stakeholders?
- Verification: – "Are we building the system right?"
- Does our design meet the spec? – Does our implementation meet the spec?
- Does the delivered system do what we said it would do?
- Are our requirements models consistent with one another?
- During analysis and design, test cases can be prepared. As the system is built, these test cases are executed to detect the presence of bugs in the programs. Use Cases describe the functional requirements of the system, and therefore they can be used to prepare test cases. Every scenario will have minimum one test case and maximum any number of test cases.

1.12 Use Case Realization

- A use case realization provides a construct to organize artifacts which shown how the physical design of a system supports the logical business behavior outlined by ause case.
- Each use case realization will define the physical design in terms of classes and collaborating objects which support the use case.