CH-6

Object Oriented Analysis process: Identifying use cases.

Analysis:

Main objective → to capture complete, and consistent picture of requirements of the system.
&
what the system must do to satisfy the user's requirements and needs.

→ Analysis is a process of transforming a problm into system requirement.
→ Analysis involves a great deal of interaction with the people who will be affected by the system.
→ Analyst has four tools for extracting information about a system.

✓ 1.) Examination of existing system documentation
✓ 2) Interviews
✓ 3.) Questionnaire
✓ 4) Observation.

why Analysis is a Difficult Activity:-

Norman → states 3 most common source of requirement difficulties

1.) Fuzzy descriptions (something not clear)
2.) Incomplete requirements
3.) Unnecessary features.

Business object Analysis: Understanding the Business layer:

→ Main objective: Understand user's requirements.
Outcome: Identify classes that make up the business layer and relationship.

⇒ To understand user-requirement, we create use cases.

(X.) → use cases are scenarios for understanding system requirement.

Use-Case driven Object-Oriented Analysis: The Unified Approach:

The O-O Analysis phase uses Actors & use-cases to describe the system from user perspective.

Actors → External factors that interact with system.

Use-cases → Scenarios that describe how actors use the system.

Steps in OOA:

1.) Identify actors
2.) Develop business process model using UML activity diagram.
3.) Develop use case
4.) Prepare interaction diagram
5.) UML class diagram
6.) Iterate & refine.

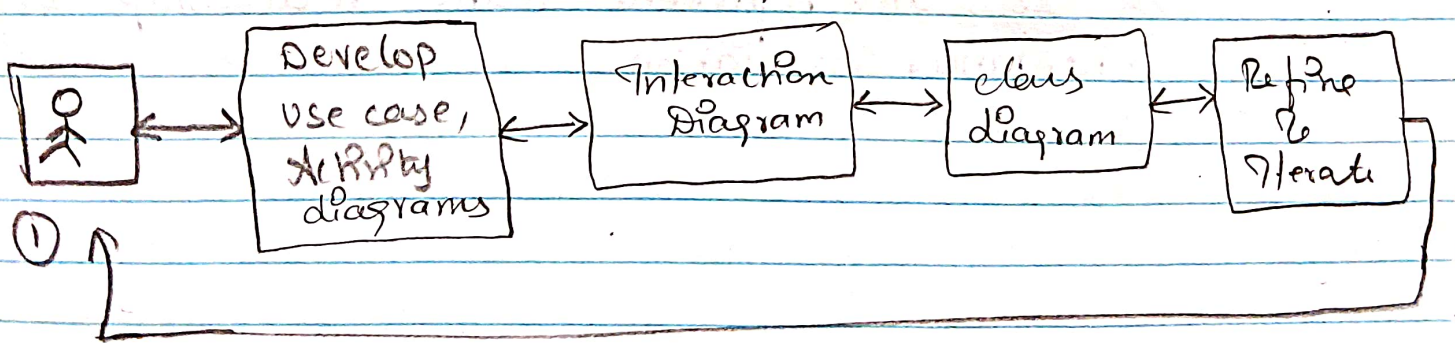


Fig: 6.1

Business process modeling:

→ not necessary for all projects.
→ when required Business process and requirements can be modeled to any level of detail.
(7.) → Activity diagram supports this modeling
→ Advantage: Familiarity.
→ Disadvantage: Time consuming process.

Fig: 6.2

Use-Case model:-

→ Use-case are scenarios for understanding System requirement.
→ Use-case is an interaction b/w user & system.
→ It captures goal of users & responsibility of the system to its users.
(8.) → It can discover classes and relationship among subsystems of the system.
(9.) → Use-case provides external view of the system.
→ Use-case model expresses what the business or application will do and not how.
→ UML-class diagram also called object model represent static relationship b/w objects, inheritance, association.

Use-case model
↓
represents the external view of the system

represents internal view of the system.
↓
class diagram

ref. Fig: 6.3

1) Use-Cases under the Microscope:-

Important issue in building correct use-case is the difference b/w user goals and system interactions.

Use cases => represent the thing that user is doing with the system.

Def: By Jacobson: "A use case is a sequence of transactions in a system whose task is to yield results of measurable values to an individual actor of the system".

Use Case:- → Flow of events through the system.

(H:) → To reduce the complexity ~~& its reduces the numbers of~~, one must group the courses of events and call each group a use-case.

→ So this grouping can reduce the number of use-cases in a package.

Actors: → It is a user playing a role in a system.

→ Actors carry out the use cases.

→ A single actor can perform many use case.

→ A use case can have several actors acting on it

→ A actor can be also an external system that needs information from current system.

→ Actors => can get value from use case (or) just participate in use-case.

In a System:

↳ Actors communicate with the system use-case.

A measurable value:

Use case must help the actor to perform a task that has some identifiable value. (Price or cost)

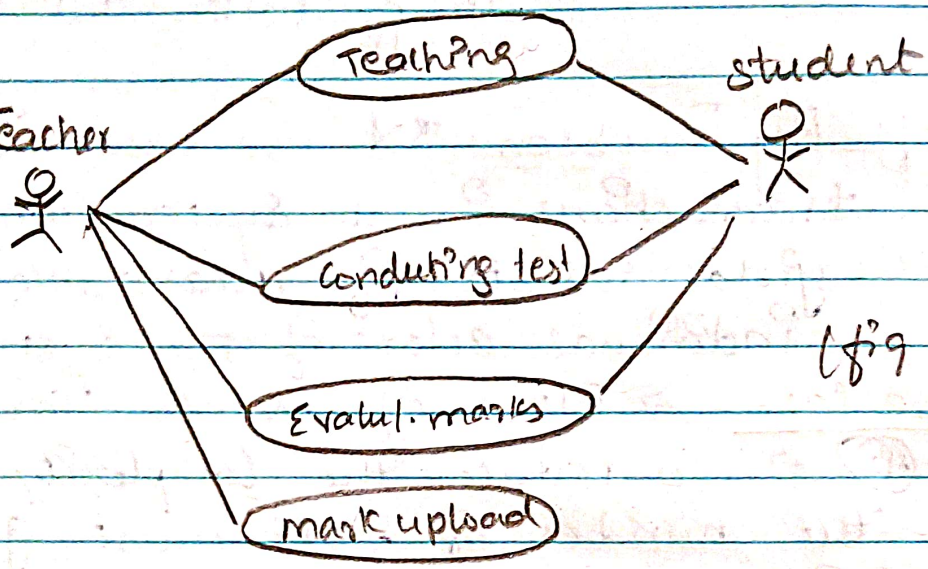
Transaction:-

→ It is an atomic set of activities that are performed fully or not at all.

→ Transaction is triggered from the actor to the system

→ Each transaction has a name.

Ex:



(fig: 6.4)

Uses and extends Association:

→ It is difficult to understand the use case when it has many flows of events.

→ so, this is eliminated by extends and uses association.

Extends association:

→ It is used when you have use case that is similar to another use case but does a bit more or is more specialized.

→ It is like a Subclass.

Uses association:

→ It is used when the use cases have subflows in common.

→ to avoid of using same subflow many times we can extract the common subflow & make it a use case of its own.

→ This new use case can be shared by other use cases.
→ This relation among other use cases and this new extracted use case is called as uses association.

Similarity of Uses & Extends:
→ Both are a kind of Inheritance.
→ Common sequence in several use case => Uses asso. (when we need to share)
→ when you add bit more specialization => Extends asso.

Abstract & Concrete use-case:-
Abstract:→ It is not complete & has no initiation actors
→ But used by concrete usecase,
→ It also has extends & uses association.
Concrete: → It interacts with actors.

Fowler and Scott: Guidelines for addressing variations in use-case modeling
1.) Capture the simple and normal use case first
2.) For every step
↳ what could go wrong?
↳ How this work out differently?
3.) Extract common sequence.

3.) Identifying the actors:
→ The term "actor" represents the role a user plays with respect to the system.
→ Cause & Weinberg explain the concept of railroad Paradox.
→ Says railroad paradox appears everywhere there are products. and
1.) The product is not satisfying the user
2.) Since product is not satisfactory, Potential users will not use it.

2.) Potential users ask for a better product.
4.) Since the potential users do not use the product, the product is denied.

Actors can be found through the foll. Q & A :-
1.) who is using the system
2.) who affects the system
3.) which H/w or external functions makes the system to perform task?
4.) what problem does this application solve?
5.) How do user's use the system?

→ Jacobson Provide two-three rule for identifying actors.
↳ Start with naming at least two, Preferably three People that loved act as a actor.
↳ other actors can be identified in subsequent iterations.

4.) Guidelines for finding use-cases :-
Steps:
1.) For each actor, find the task & functions, that the actor should be able to perform or that the system needs the actor to perform.
2.) Name the use cases.
3.) Describe the use case briefly with which the user is familiar.

→ Actors represent a role that users can play.

5.) Use-case: when to stop decomposing & when to continue:-

→ During analysis of a business system, one can develop one use-case diagram.

→ The extends and uses relationship can be used to eliminate redundancy.

→ capturing use-cases are the primary task of analysis phase.

→ when we have arrived at the lowest use-case model, which cannot be broken down any further, we can create Sequence, Collaboration diagram.

6. Dividing use cases in to Packages:- fig 6.6

7. Naming a use case:

→ use case name should provide general description of use-case function.

Developing Effective Documentation:-

→ Documentation often helps reveal issues and gaps in the analysis and design.

→ Documentation can be an important factor in making decision

Organization Conventions for Documentation:-

→ Documentation depends on organizations rules and regulations.

→ Many organizations est. standards for developing documents.

Guidelines for developing effective documentation:-

→ Bell and Evans provides guidelines and template for preparing a document.

1.) Common Cover: Common cover sheet that identifies the document (Includes version, person responsible) etc.,
2.) 80:20 rule: 80% of work to be done with 20% of documentation.
3.) Familiar Vocabulary:- Use simple terms.
4.) Make document as short as possible
5.) Organize the document

CH-7

Object Analysis : Classification

Def : OO Analysis is a process by which we can identify classes.

→ Identification of classes is the hardest part of OOA & Design.

Classification Theory :-

→ It's a process of checking that an object belongs to a category or class.

→ classification plays a role in allocating processes to procedures.

Class are an important mechanism for classifying objects.

The chief role of class is to define the attributes, methods & instances.

classes are important bcz they create conceptual building blocks for designing systems.

Approaches for identifying classes:-

1.) Noun Phrase approach
2.) Common class pattern approach
3.) Use case driven approach
4.) classes, Responsibilities & Collaborators.

1.) Noun Phrase approach:-

Nouns are divided in to
1.) Relevant classes
2.) Fuzzy classes
3.) Irrelavant classes.

Selecting clauses Identifying Tentative clauses :-

1.) Look for nouns & noun phrases
2.) Some clauses are implicit or taken from general knowledge.
3.) Clauses must be in application domain.
4.) Carefully choose & define class names.

Selecting clauses from Relevant and Fuzzy categories :-

1.) Redundant clauses :- Do not keep two clauses that expres the same information, so select the one that is more meaningful in context.
2.) Adjective clauses :- If the object represented by a noun behaves differently when adjective is applied, Then we can create new class.
3.) Attribute clauses :- If object have the capability to be defined as attribute then no class should be created. (It should not be created as class).
4.) Irrelevant clauses :- If the class does not have a purpose & necessary, we can simply eliminate the candidate class.

2.) Common class pattern approach :-

Methods / Patterns for finding the candidate class.

1.) Concept class :- when an idea is shared by another, it becomes a concept. It's used to keep track of business activities or communications.
2.) Event class :- Things happen in time (usually at give time, date, order) must be recorded.
3.) Organization class :- Collection of people, resources, facilities or groups.

4.) People class :- Different role user plays in interacting with the application. Persons can be divided in to two types
↳ 1.) users of the system
↳ 2.) People who do not use the system but about whom infol. is kept at the system.

5.) Places class :-
Physical location.

6.) Tangible things and devices class :-
Physical object or group of object with which the application interacts.

3.) Use-Case Driven Approach :-
Use case modeling is considered a problem-driven approach to OOA.
↳ sequence diagram
↳ colloboration diagram.

4.) Classes, Responsibilities and Collaborators :-
→ CRC is a technique used for identifying classes, Responsibilities and attributes & methods.
→ CRC is based on the idea that object has certain responsibility itself or it requires the assistance of other objects.
→ CRC card are 4" x 6" index cards.
→ All the infol. of the object is written on a card.

Classname Responsibilities	Collaborators ....

Naming classes :- (Guidelines)

1.) Class name should be singular.
2.) Use-names with which the users / clients are comfortable.
3.) Class name should reflect its nature.
4.) Capitalize class names.
5.) Class names should begin with an upper-case.

CH-8

Identifying object Relationship, Attributes and Methods.

Introduction:-

→ object donot exist in isolation, but interacts with each other.

→ These interactions and relationship are the application.

→ 3 types of relationship among objects

1.) Association (Guide us in designing classes).
2.) Super-Sub structure (also called Generalization hierarchy) (Guide us in direction of inheritance).
3.) Aggregation and a-part-of structure

1.) Associations:-

Definition:-

→ Relationship among objects are known as associations.

→ Association represents the physical or conceptual connection b/w 2 or more objects.

Types:

1.) Binary association (b/w 2 classes)
2.) Ternary or higher order (Diamonds connecting a class).

Identifying associations:-

Questions

1.) Is the class capable of fulfilling the required task by itself?
2.) If not, what does it need?
3.) From what other class can it acquire what it needs?

Common Association pattern:-

1.) Location association → next to, Part of, contained in.
2.) Communication association → talk to, order to.

Elimination of unnecessary association:-

1.) Implementation association:-
Concerned with implementation or design of a class & not relationship among business object.

2.) Ternary associations:-
Relationship among more than 2 classes. It is complicated, so convert to binary association.

3.) Directed actions:-
→ Derived associations (also called)
→ They are redundant, so avoid this.

) Super-Sub class relationships:-
→ It represent Inheritance relationship blw related classes & class hierarchy determines the lines of Inheritance blw classes.
→ Also called as Generalization hierarchy, allows objects to build from other objects.

Guidelines for identifying super-sub Relationship:-

1.) Top-down:-
→ Looking for adjectives (same word can give diff. meaning)
→ Can discover additional special cases.

2.) Bottom-up:-
→ Looking for classes with similar attributes or methods.
→ Moving the common attributes & methods to abstract class.

3.) Reusability →
→ Do not create very specialized classes at the top of hierarchy.

a.) Multiple Inheritance:-

→ Avoid excessive use of multiple Inheritance.

→ One way of achieving multiple Inheritance is to Inherit from the most appropriate class.

a.) A-part-of Relationship - Aggregation:-

→ It represent a situation where a class consist of several component classes.

→ 2 major properties.

1.) Transitivity:-

A → B → C (i.e) A → C

2.) Antisymmetry:-

A → B But B $\not\rightarrow$ A

→ Hollow or filled diamond is used to represent aggregations.

→ Filled diamond signifies strong form of aggregation which is composition.

4.) Class Responsibility: Identifying Attributes and methods.

Use cases and UML diagrams is used to identify attributes, methods, relationship among classes.

Guidelines for defining attributes:-

1.) Attributes corresponds to nouns, adjectives or adverb.

2.) State only enough attribute, by keeping class simple.

3.) Attributes are not fully described in prblm so need knowledge of application domain and real world to find them.

4.) Omit derived attributes.
5.) Do not carry discovery of attributes to excess.

Object Responsibility Methods and Messages :-
→ Methods and messages are workhorses of object-oriented system.
→ Every object is surrounded by rich set of methods.