OOADL :-

OOAD :- Introduction to UML.

Iterative, Evolutionary and Agile : Intro to Unified process, Agile Modeling, Agile Unified process.

Inception & Use cases : Intro, Evolutionary Requirements, Use cases, Use case Diagrams, Activity Diagram.

OOAD : It is a software engineering methodology that involves use of OO concepts to design and implement software systems. It involves a number of techniques and practices including :

- Object-orient programming.
- design patterns
- UML diagrams and use cases.

1) Object-oriented programing : involves modeling real-world objects as software objects, with properties and methods that represent the behaviour of those objects. (to design & implement software systems).

2) Design patterns : are reusable solutions to common problems in software design. OOAD uses design patterns (to help developers create more maintainable and efficient software systems.)

3) UML Diagrams : standardized notation for creating diagrams that represent different aspects of a software system. (to represent different components and interactions of a software system).

4) Use cases : are a way of describing the different ways in which users interact with a software system. (to help developers understand the requirements of a system and to design software systems that meet those requirements).

Advantages :	Disadvantages :
* Reusability.	* Complexity.
* Scalability.	* Time-consuming.
* Maintainability.	* Rigidity.
* Flexibility.	* Cost

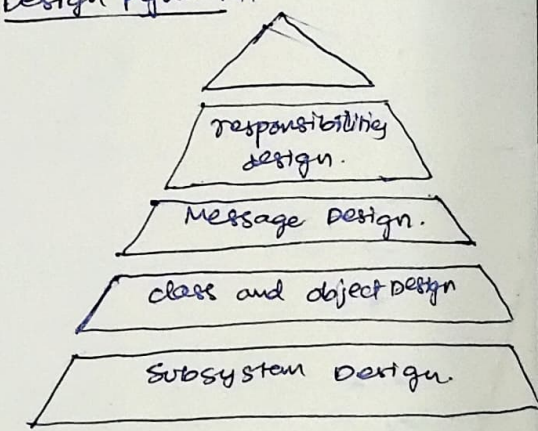
Object-Orient Analysis : introduces new concepts to investigate a problem. It is based on a set of basic principles which are as follows -

1. The information domain is modeled.
2. Behavior is represented.
3. The function is described.
4. Data, Functional, and behavioral models are divided to uncover greater detail.
5. Early models represent the essence of problem, while later ones provide implementation details.

Object-oriented design : An analysis model created using OOA is transformed by OOD into a design model that works as a plan for software creation.

* It contains several different levels of modularity.
* The major system components are partitioned into subsystems, and data manipulation operations are encapsulated into objects.

Design pyramid :



1. Subsystem Layer : Subsystem that enables software to achieve user requirements and implement technical framework that meet user needs.
2. Class and Object Layer : represents the class hierarchies that enable the system to develop using generalization and specialization.
3. Message layer : represents the design details that enable each object to communicate with its patterns. establishes internal and external interfaces for the system.
4. Responsibilities Layer : represents data structure and algorithmic design for all attributes and operations.

EXPT. No.
Date

Object Oriented Concepts Terminologies

1. Attributes: a collection of data values that describe a class.
2. Class: encapsulate the data and procedural abstractions required to describe the content and behaviour of some real-world entity.
   generalized description that describes the collection of similar objects.
3. Objects: Instances of a specific class. They inherit a class's attributes and operations.
4. Operations: also called methods & services, provide a representation of one of the behaviors of the class.
5. Subclass: specialization of super class. A subclass can inherit both attributes and operations from a super class.
6. Superclass: also called a base class, generalization of a set of classes that are related to it.

Advantages

1. Improved modularity
2. Better abstraction.
3. Improved reuse.
4. Improved communication.
5. Reusability.
6. Scalability.
7. Maintainability.
8. Flexibility.
9. Improved software quality.

Disadvantages

1. Complexity
2. Overhead
3. Steep learning curve.
4. Time consuming
5. Rigidity.
6. Need expertise
7. Cost.

Introduction to UML:

* Unified Modeling Language (UML) is a modeling language that can be used for a variety of purposes.
* primary goal of UML is to establish a standard way to visualize how a system has been constructed.

What is UML?

* Also called graphical modeling language.
* It specifies, visualizes, builds and documents the software system's artifacts (main elements).
* UML diagrams are used to depict the behavior and structure of a system.
* UML facilitates modeling, design and analysis for software developers, business owners, and system architects.

Benefits:-

* Straight forward
* can be used by any modeler.
* easy to implement.
* Scalable.
* easier to understand.
* Save's time in the long run.

characteristics:

* generalized for various use cases.
* not programming language; instead, it is a graphical modeling language.
* close connection to OOAD.
* used to visualize system's workflow.

Conceptual Modeling:

* depicts the business domain at a high level, utilizing language and concepts that business users are familiar with while neglecting logical and physical level aspects (underlying DB or OS) and external

Page No.

level aspects (screen forms used for data entry).

* A conceptual model is made up of multiple interconnected concepts.
* It simplifies the understanding of the objects and how they interact with one another.
* This is the preliminary stage before creating UML diagrams.

Object: real world entity.

Class: key construct, group of objects.

Abstraction: conveying to users the essential properties of an object while suppressing extraneous information. It is used to imagine how an object will function.

Inheritance: process of creating a new class from an existing one. New elements and methods can be added to the new class, but the parent class's elements and methods are available to objects in the child class without having to rewrite their declarations.

polymorphism: mechanism for expressing objects with various forms that can be utilized for different reasons.

Encapsulation: It joins the data and the methods as a single entity, allowing tight coupling.

Goals of OOAD:

* Identifying a system's objects.
* Identifying their connections.
* Creating a design that can be translated into executables utilizing OO languages.

Role of UML:

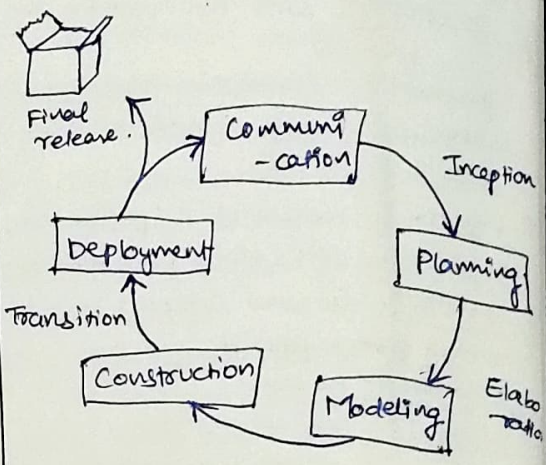
1. Requirement Analysis & specification.
2. System Design & Architecture.
3. Behavioral Modeling.
4. Code Generation & Implementation.
5. System validation & Testing.
6. Communication among stakeholders.
7. Scalability & Maintainability.

Unified Process

The Unified Process (UP) is a software development framework used for object-oriented modeling. The framework is also known as Rational Unified Process (RUP) and the Open Unified process (Open UP).

Key features:

* It defines the order of phases.
* It is component based.



* It follows an iterative, incremental architecture-centric, and use-case driven approach.

Case-Driven Approach:

* follows a set of actions performed by one or more entities.
* A use case refers to the process of the team performing the development work from the functional requirements.

Architecture centric approach:

* defines the form of the system and how it should be structured to provide a specific functionality.

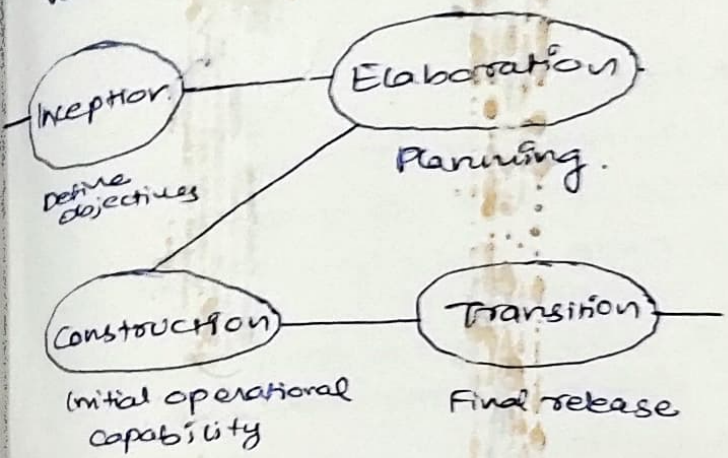
usecase → define → functionality

Iterative & Incremental approach:

* the product will be developed in multiple phases.
* developers evaluate and test.

Phases

* series of cycle.
* Each cycle ends with the release of a new system version for the customers.



Inception :

* delimiting the project scope.
* What are the key features ?
* benefit the customers ?
* Methodology ?
* risks involved ?
* schedule and cost estimates.

Elaboration :

* build the system given the requirements, cost, and time constraints and all the risks involved.
* Develop with the majority of the functional requirements implemented.
* Finalize the methodology.
* Deal the risks.

Construction

* development, integration and testing
* complete architecture is built here.
* hand the final documentation to the client.

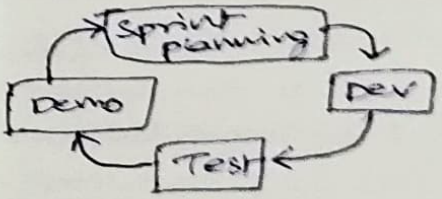
Transition

* involves the deployment, multiple iterations, beta releases and improvements of the software.
* Users will test the software which may raise potential issues.

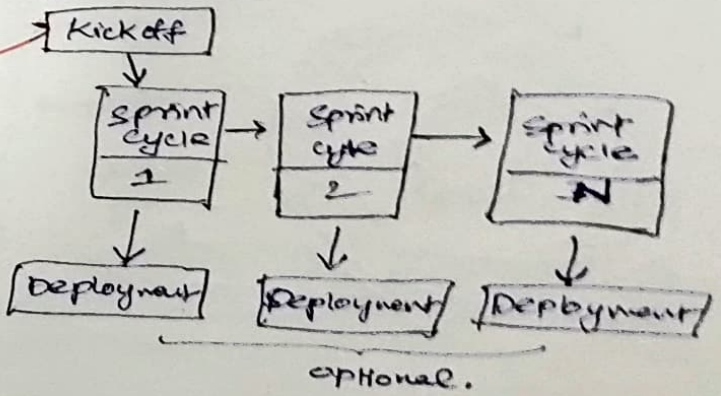
Agile Modeling

* type of incremental model.
* Software is developed in incremental, rapid cycles.
* Results in small incremental releases with each release building on previous functionality.
* Each release is thoroughly tested to ensure software quality is maintained.
* used for time critical applications.
* Extreme programming (XP) is currently one of the most well known agile development life cycle models.

Sprint cycle :



Agile model :



Advantages:

* Rapid & continuous delivery (customer satisfaction).
* constant interactions
* Working software is delivered frequently.
* Communication (1:1).
* Continuous attention.
* Regular adaptation.

Disadvantages:

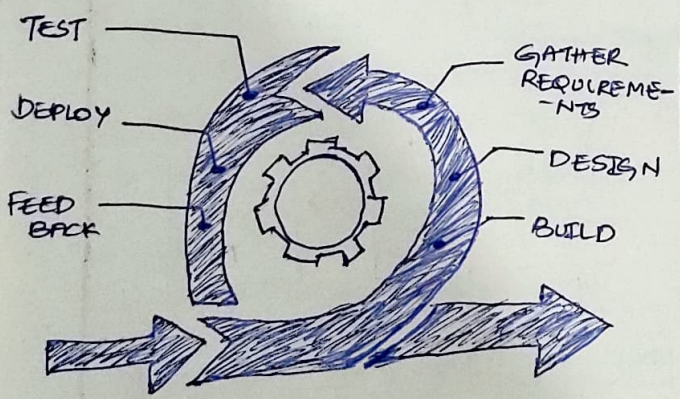
* Difficult for large scale projects.
* Lack of emphasis
* customer representation should be clear (project offtrack's),
* Only senior programmers.

When to use?

* When new changes are needed to be implemented. (very little cost).
* Less time to implement new feature.
* Unlike waterfall, agile model very limited planning is required.
* constant feedback from client.
* rigid sequential way (for both system developers & stakeholders).

Phases of Agile Model:

* Agile breaks down tasks into smaller iterations.
* Each of which lasts for a short time frame (one to four weeks) in the overall process model.



Requirements Gathering:

⇒ Plan ⇒ Explore ⇒ Evaluate.

* define the projects requirements.
* includes explaining business opportunities and planning the time and effort required for the project.
* Once information is quantified, technical & economical feasibility of the project is evaluated.

Design:-

⇒ Refine ⇒ collaborate

⇒ Stakeholder Buy-in

* Once project parameters are identified.
* Work with the stakeholders to define the requirements.

Build:-

⇒ Iterate ⇒ Develop ⇒ improve.

* product, design, and develop teams start working on related projects, ultimately deploying a product or service that is not static.

Test:-

⇒ Debugging ⇒ Reduce waste

⇒ Quality assurance.

* Quality Assurance (QA) team examines and evaluates the product's performance, looking for bugs and other flaws.

Deploy:-

⇒ Iterate ⇒ Automated testing.

⇒ Continuous integration.

* The team deploys the product in a working environment.

Feedback:

⇒ Issue resolution ⇒ User interaction

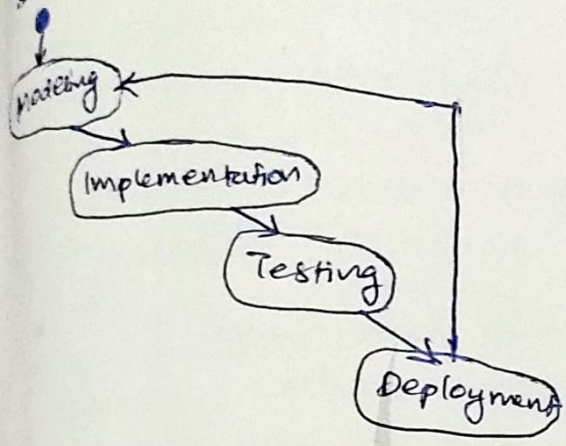
⇒ continuous improvement

* The team receives feedback about the product and handles any issues that may have arisen.

EXPT. No.
Date

Agile Unified Process (AUP)

* Agile version of the Rational Unified Process (RUP).
* It is an iterative-incremental process consisting of four sub processes or workflows:



* It combines (Business Modeling, Requirements, Analysis and Design) into an agile modeling workflow.
* Agile modeling produces minimalist throw-away models.
* AUP defines the same three supporting workflows as RUP:
  - → Project Management
  - → Configuration
  - → Management
  - → Environment.
* AUP iterations are required to be short (abt two weeks).

Agile Methodology:

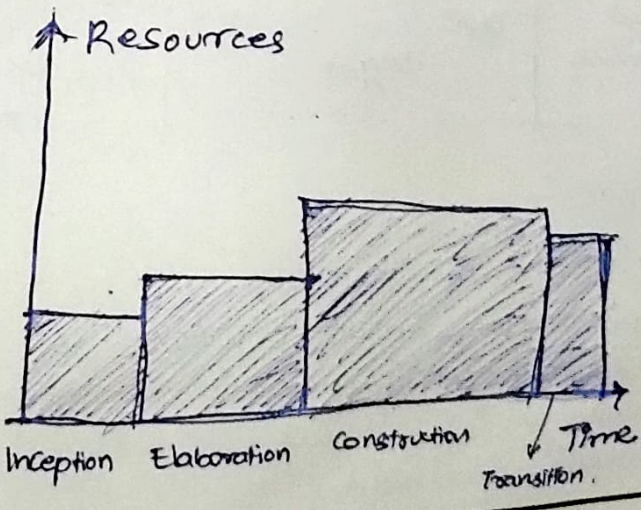
* It is an alternative to traditional approaches that focus on one major deliverable.
* Agile breaks down goals into independent products which can be developed and released quickly without having any long-term consequences
* can be quite simple with the two main styles being Scrum and Kanban.
  - Kanban → Service Request Manager (SRM).
    - ↳ Delivery Specialist.
  - Scrum → 5-6 people.
    - ↳ Product Owner
    - ↳ development leader
    - ↳ team member.
* Agile workflow is a flexible system of work with short task spans that are often called Sprints.

Agile's Core Principles:

* Values pragmatism over planning.
* allowing for more flexibility throughout the process.

Agile, in it's purest form, emphasizes four key principles:

⇒ Personal engagement over processes and tools.
⇒ Flexibility.
⇒ Functional Software.
⇒ Collaboration with customer.



Agile	Unified process (UP)
Iterative & Incremental	Iterative & Incremental
Highly flexible, adaptable to changes.	Structured with predefined phases.
No strict phases; continuous iterations	Four phases (IECT).
Minimal documentation, focuses on working software.	Emphasizes documentation and models.
Continuous collaboration with customers	Customer involvement mainly in early & final stages
Adaptive planning, responds to changes.	Defined planning based on milestones
Small to medium sized projects.	Large, complex project

↓ Speed & flexible | Both can be used together. | ↓ focuses on structure

F Inception:

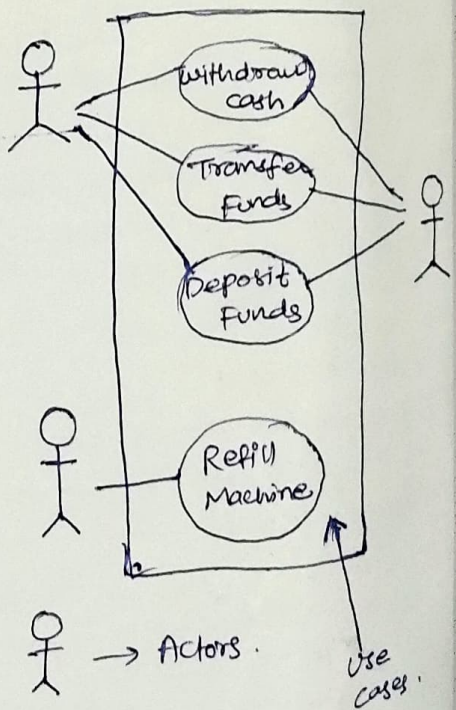
Activities:

* Formulate the scope of the project.
* Needs of every stakeholder, scope, boundary conditions and acceptance criteria established.
* Plan and prepares the business case.
* defines risk mitigation strategy, develop an initial project plan and identify known cost, schedule, and profitability trade-offs.
* Synthesize candidate architecture.

Exit criteria:

* An initial business case containing at least a clear formulation of the product vision.
* The core requirements:
  - ↳ functionality.
  - ↳ scope.
  - ↳ performance.
  - ↳ capacity
  - ↳ technology base.

ATM Elaboration Usecase Diagram:



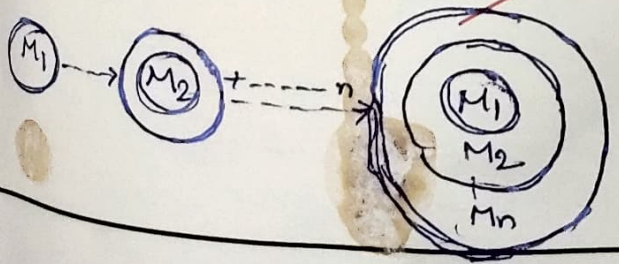
EXPT. No.
Date

Evolutionary Requirements

* Requirements are capabilities and conditions to which the system - and more broadly. The UP promotes a set of best practices, one of which is manage requirements.
* In the context of inevitably changing and unclear stakeholder's wishes
* A systematic approach to finding, documenting, organizing and tracking the changing requirements of a system".
* Iteratively and skillfully and not being sloppy.
* prime challenge of requirements analysis is to find, communicate, and remember what is really needed.

Evolutionary Development

* Software development approach that is based on incremental and iterative development of software systems.
* Adaptive approach that focuses on delivering working software early and then refining it over time based on feedback from users and stakeholders.



* Based on a sequential and linear process, where the requirements are defined upfront.
* Design and implementation of the system follow a predetermined plan.
* The design of the system is broken down into small, manageable pieces, and each piece is developed and tested seperately.

ED Process:

1. Requirements Gathering.
2. Design.
3. Implementation.
4. Testing.
5. Feedback and Refinement.
6. Maintenance.

Benefits:

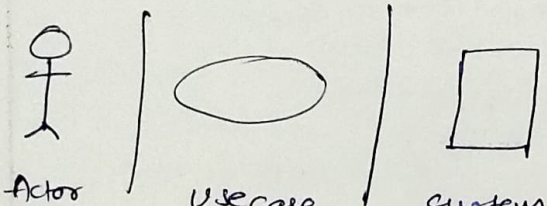
1. Flexibility.
2. Easly Delivery.
3. Cost savings.
4. Improved Quality.
5. Customer satisfaction.

Drawbacks:

1. Lack of Documentation.
2. Scope creep.
3. Integration Issues.
4. Technical Debt.
5. Limited Reusability.

Use-case Diagram

* referred to as a Behaviour model (or) diagram.
* Simply describes and displays the relationship (or) interaction between the users (or) customers and providers of the application service or the system.
* Describes the different actions that a system performs in collaboration to achieve something with one or more users of the system.
* In UML, It represents the interaction between actors and a system. provides a high-level view of the system's functionality.



Actor
* external entities that interact with the system.
* Actors initiate use cases and receives the outcomes.

Usecase
* Scenes in the play.
* represent specific things the system can do.

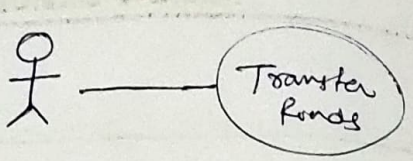
System Boundary.
* visual representation of the scope (or) limits of the system you are modelling.
* defines what is inside the system and what is outside.

Relationships:

1. Association Relationship:

* represents a communication (or) interaction between an actor and a use case.
* Depicted by a line connecting the actor to the use case.

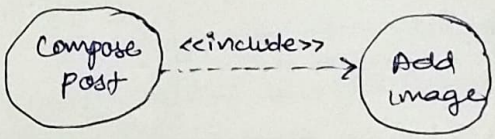
Ex:-



2. Include Relationship.

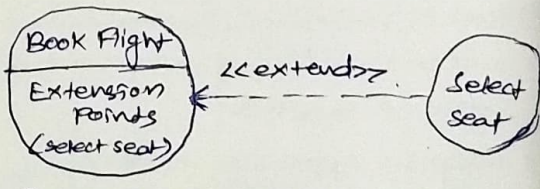
* Indicate that a use case includes the functionality of another use case.
* denoted by a dashed arrow pointing from the including use case to the included use case.
* It promotes modular and reusable design.

Ex:-



3. Extend Relationship:

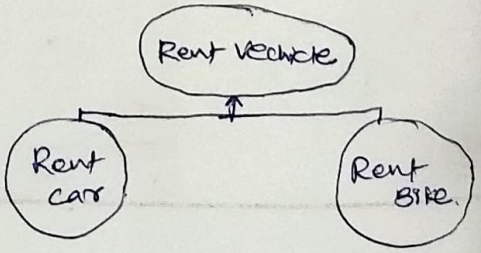
* illustrates that a use case can be extended by another use case under specific conditions.
* is useful for handling optional (or) exceptional behaviour.



4. Generalization Relationship:

* establishes an "is-a" connection between two use cases, indicating that one use case is a specialized version of another.

Ex:-

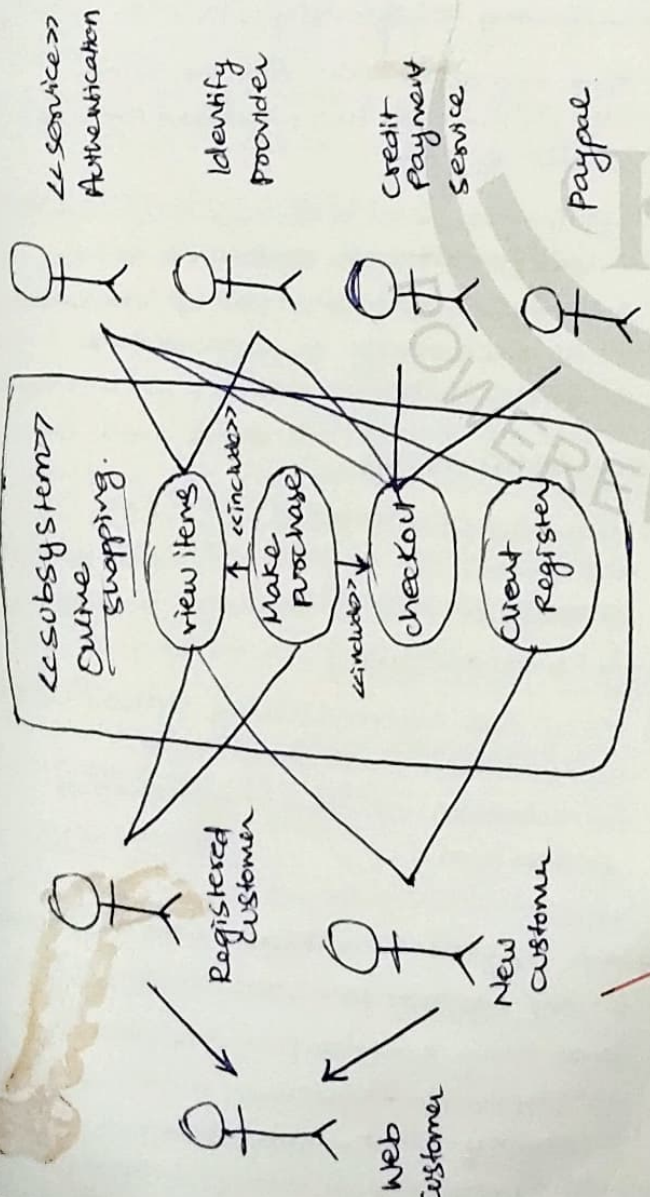


Date

Steps to draw use case diagram:

1. Identify Actors.
2. Identify use cases.
3. connect Actors & usecases
4. Add System Boundary.
5. Define relationships.
6. Review and Refine
7. validate.

Ex:- Online Shopping System.



Activity Diagram

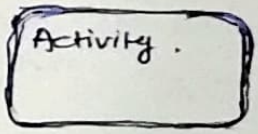
* Activity, usecase & State-machine diagrams are considered behavior diagrams because they describe what must happen in the system being modeled.
* They help people on the business and development sides of an organization come together to understand the same process & behavior.

Benefits

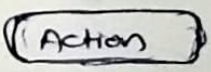
* Demonstrate the logic of an algorithm.
* Describe the steps performed in a UML usecase.
* illustrate a business process (or) workflow between users and the system.
* Simplify and improve any process.
* architecture elements.

Diagram Notation:

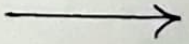
* **Activity**
  (represent set of action)



* **Action**
  (a task to be performed)



* **Control Flow**
  (shows the sequence of execution)



* **Object Flow**
  (shows the flow of an object from one activity to another)

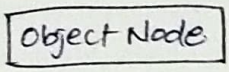
* Initial Node

* **Activity Final Node**

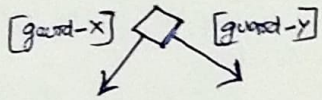
(Stop all control flow and object flows in an activity).

* **Object Node**

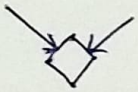
(represent an object).



* **Decision Node**

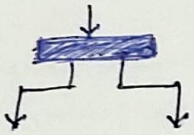


* **Merge Node**



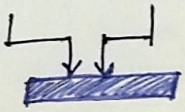
* **Fork Node**

(Split behavior into a set of parallel (or) concurrent flows of activities)



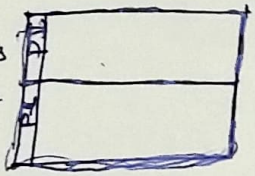
* **Join Node**

(Bring back together)

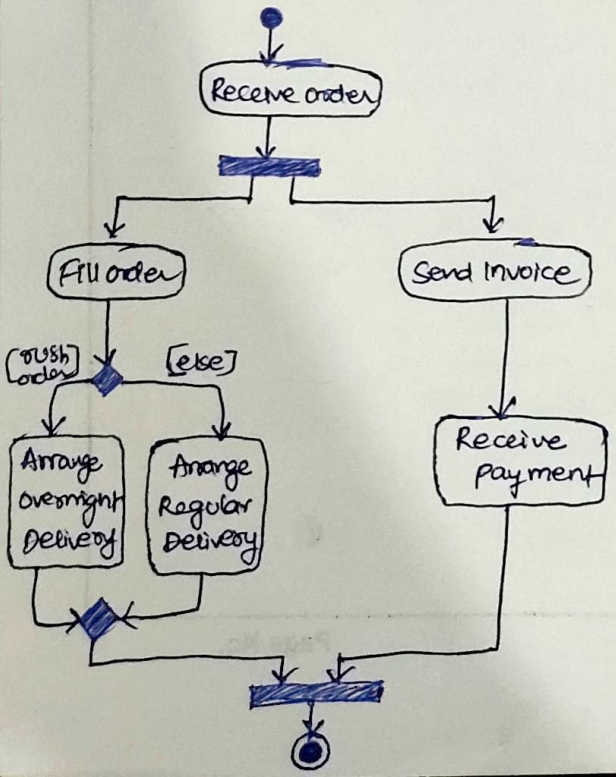


* **Swimlane & Partition**

(A way to group activities performed by the same actor on an activity)



Ex:- Process Order (Activity Diagram).



OOAD 2 :

Elaboration & Domain Models: Iteration Requirements and Emphasis: Core OOAD skills, Domain Models, class Diagrams, System sequence Diagrams, Requirements to Design,

Package Diagram and UML Tools: Logical Architecture, software Architecture, Package Diagrams, on to object Design, UML CASE tools, UML class Diagrams.

Iteration 1; Requirements and Emphasis; Fundamental OOAD skills

* emphasizes a range of fundamental and common OOAD skills used in building object systems, such as assigning responsibilities to objects.
* The requirements for the first iteration of the NextGen POS application follow:
  - ↳ Implement a basic, key scenario of the Process Sale use case.
  - ↳ Implement a Start Up use case as necessary to support the initialization needs.

Core OOAD Skills:

* The object model visualizes the elements in a software application in terms of objects.

Objects and Classes:

* They are intrinsically linked with each other and form the foundation of object-oriented paradigm.

Object:

* An object is a real-world element in an object-oriented environment that may have a physical or a conceptual existence.
  - ↳ Identity that distinguishes it from other objects in the system.
  - ↳ determines the characteristic properties of an object as well as the value of the properties.

* Behavior that represents externally visible activities
* can be modeled according to the needs of the application.
* May have a physical existence (or) an intangible conceptual existence.

Class:

* represents a collection of objects having same characteristic properties that exhibit common behavior.
* It gives the blue print (or) description of the objects that can be created from it.
* creation of an object as a member of a class is called instantiation.

Constituents of a class:

* A set of attributes for the objects that are to be instantiated from the class.
* Generally, different objects of a class have some difference in the values of the attributes.
* Attributes are often referred as class data.
* A set of operations that portray the behavior of the objects of the class.
* Operations are also referred as functions (or) methods.

Encapsulation:

* the process of binding both attributes and methods together within a class.
* The internal details of a class can be hidden from outside.
* It permits the elements of the class to be accessed from outside only through the interface provided by class.

Data Hiding:

* A class is designed such that its data (attributes) can be accessed only by its class methods and insulated from direct outside access.

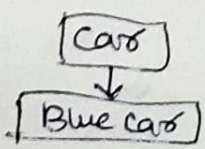
Message Passing:

* Any Application requires a number of objects interacting in a harmonious manner.
* objects in a system may communicate with each other using message passing.

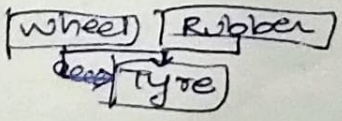
Inheritance: (is-a).

* Mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities.
* Existing classes also → Base / parent / super classes.
  New classes also → derived / child / sub classes.
* Sub class may add its own attributes and methods and may modify any of the super class methods.

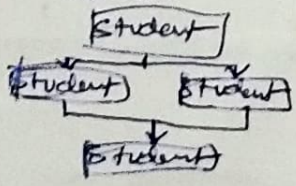
Single Inheritance



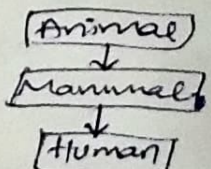
Multiple

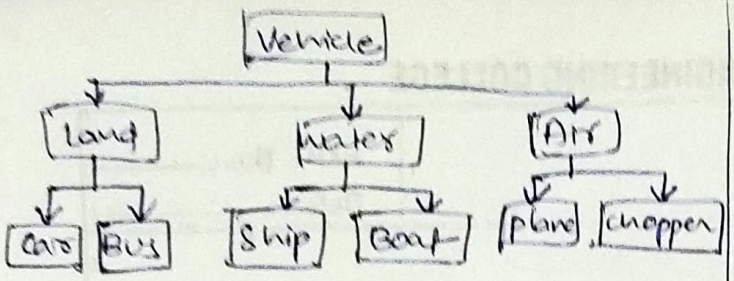


Hybrid



Multi-level





Hierarchical

Polymorphism:

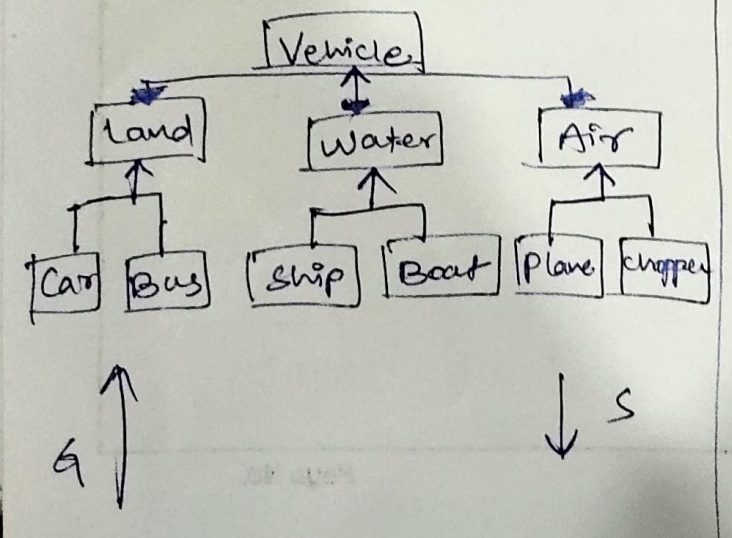
* that means the ability to take multiple forms.
* polymorphism implies using operations in different ways, depending upon the instance they are operating upon.
* It allows objects with different internal structures to have a common external interface.

Generalization

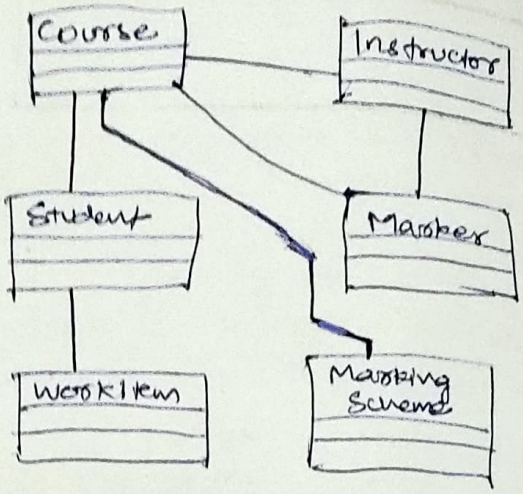
* The common characteristics of classes are combined to form a class in a higher level of hierarchy.
  ie., Subclasses are combined to form a class in a ~~higher level of hierarchy~~ generalized superclass.

Specialization

* reverse process of generalization.
* distinguishing features of groups of objects are used to form specialized classes from existing classes.
* Subclasses are the specialized versions of the super-class.



Example : GRADE REPORT Class diagram



System Sequence Diagrams

Sequence Diagrams: type of UML diagram that visually represent the interactions between object or components in a system over time.

* They focus on the order and timing of messages or events exchanged between different system elements.
* The diagram captures how objects communicate with each other through a series of messages, providing a clear view of the sequence of operations/processes.

Why use?

* Visualizing Dynamic Behavior.
* Clear communication.
* Use Case Analysis.
* Designing System Architecture.
* Documentation System Behavior.
* Debugging and Troubleshooting.

Notations:-

1. Actors:-

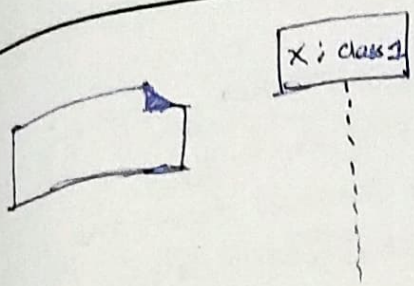
2. Lifelines:

⇒ element which depicts an individual participant in a sequence diagram.

⇒ each instance is represented by a lifeline.

⇒ are located at the top in a sequence diagram.

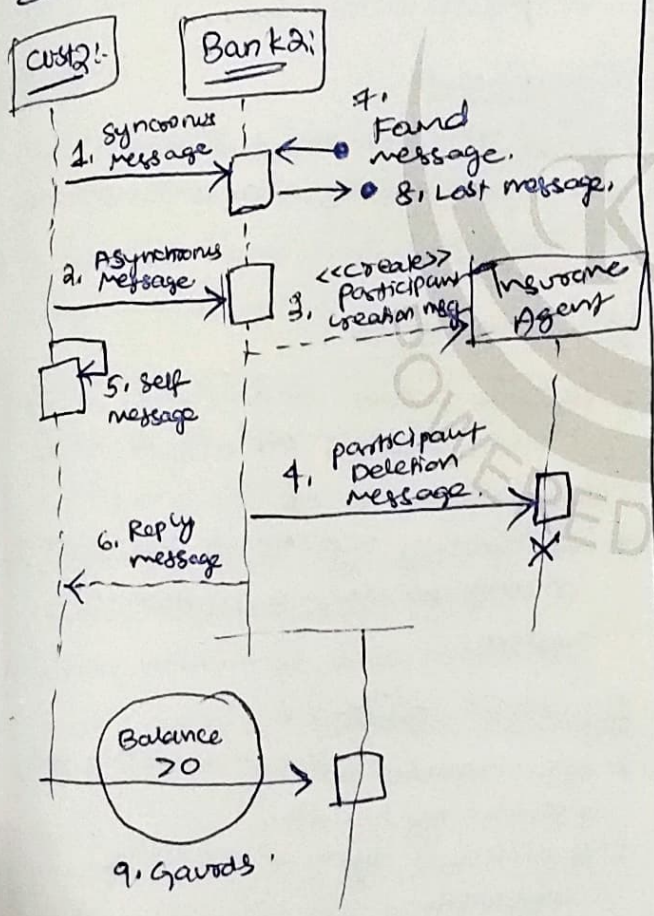
Date



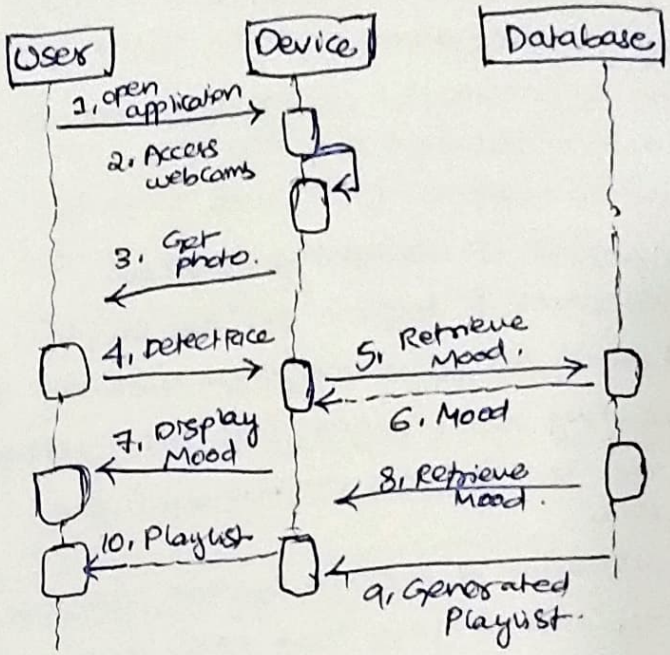
3) Messages

* communication between objects is depicted using messages.
* The messages appear in a sequential order on the timeline

Types :-



Example :- emotion based music player.



Steps involved :

1. Identify the Scenario.
2. List the participants.
3. Define lifelines.
4. Arrange Lifelines.
5. Add Activation Bars.
6. Draw Messages.
7. Include Return Messages.
8. Indicate Timing and order.
9. Include conditions and Loops.
10. consider parallel Execution.
11. Review and Refine.
12. Add Annotations and comments.
13. Document Assumption & constraints.
14. Tools.

Page No.

Introduction to Logical Architecture:

* Structural design that is used to provide a detailed description of the system without defining the system technology or environment.
* It includes all the information related logically and detailed description without using technical terms in it.
* It can be considered a diagram used to define the relation between the software components.
* Main propose of designing logical architecture is to plan for the system and use it for communication before designing and implementing the system.
* The user can get an overview of the system.
* Organization of the subsystem, software classes, and layers that make the complete logical system.
* The components are placed randomly in the system.
* The system is decomposed into smaller logical components that directly interact with the system to meet the system requirements.
* The logical components are the abstraction of physical ~~computation~~ components, which means they can be used to perform complete system functionality while taking care of the system constraints.
  Ex:- User interface in web browser.
* It acts as an intermediate between the physical architecture and the black box system.

Components of Logical Architecture:

⇒ For a system, there is one common tier architecture is defined that contains three tiers that are used to defining the response-request cycle.

1. The user uses the presentation tier to generate the request.
2. The request is immediately transferred to the application tier.
3. In application tier, further processing is done, and the data is fetched from the data tier that is required in manipulation.
4. The response is created by the application tier and transferred to the presentation tier.
5. The response is transferred to the user using the presentation tier.

Components :-

* Client
* Presentation
* Acers
* Business service
* Data.

UML case Tools:

* Used to create meaningful, object oriented models for a software application.
* It clearly represents the working of any hardware / software system.

1. Star UML.

* open source software modeling tool, provided by MKLab.
* 11 different types of modeling diagrams.
* Supports UML 2.0 specified diagrams
* Multiplatform support.

2. Umbrello:

* Unified modeling language tool, based on KDE technology.
* Supports both reverse engineering and code generation for C++ & Java
* It implements both structural and behavioral diagrams.
* It imports C++ and can export up to a wider range of languages.

Date

3. UML Designer Tool :

* helps in modifying and envisioning UML 2.5 models.
* It allows you to create all of the UML diagrams.
* It provides transparency to work on DSL.
* user can reuse the provided presentations.
* It implements Component, class and composite structure diagrams.

4. Altova :

* has provided UModel, which is another UML software modeling tool.
* supports all types of 14 UML2 diagrams as well as SysML for ES.
* generates visually designed software models by incorporating Java, C++, and C# & Visual Basic .NET.
* provides a dedicated toolbar for an individual diagram.
* offers unlimited undo/redo.
* easily adds a hyperlink to any element.

5. Umple :-

* object-oriented and modeling language that textually supports state diagrams & class diagrams.
* It adapts JAVA, C++ and PHP, which results in more readable and short lines of code.
* Includes singleton pattern, keys, immutability, mixins, and aspect oriented code injection, which makes UML more understandable.
* It enforces referential integrity by supporting UML multiplicity.

Software Architecture :

* System's blueprint, while others identify it as the roadmap for system growth.
* consists of four features
  - Software structure
  - Architecture decisions.
  - Architecture characteristics.
  - design principles.

Architecture styles used :

1. Layered Architecture.
2. Virtualized Middleware.
3. Customer user interface.
4. Monolithic user interface.
5. Core system.

Refer 2.4 page 2 Img.

Characteristics :-

* Availability
* Reliability.
* Testability
* Scalability.
* Security
* Agility.
* Fault Tolerance
* Elasticity.
* Recoverability
* Performance.
* Deployability
* Learnability.

Summary :-

1. Software Architecture: It serves as a system's blueprint, involving structure, decisions, characteristics, and design principles.
2. Architect's Role :- Includes decision making, continuous

evaluation, staying updated, ensuring compliance, broad technical exposure, business knowledge, interpersonal skills, and handling workplace politics.

3. Evolution of Architecture -

- ↳ Monolith.
- ↳ Distributed Monolith.
- ↳ Service-oriented Architecture (SOA)
- ↳ Event-Driven Microservices (EDM)
- ↳ EDM with Micro frontends

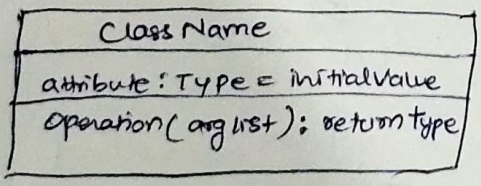
4. Architectural Design & Patterns -

- Covers layered architecture.
- UML package diagrams,
- design patterns
  - ↳ Façade
  - ↳ observer.
  - ↳ controller

5. Model-View Separation: Ensures UI and business logic remain decoupled, improving scalability and development efficiency.

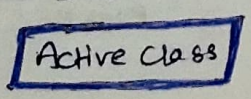
UML Class Diagram

- describes static structure of a system.
- classes represent an abstraction of entities with common characteristics.
- Associations represent the relationships between classes.



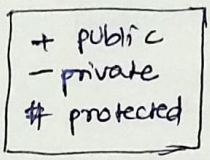
Active class:

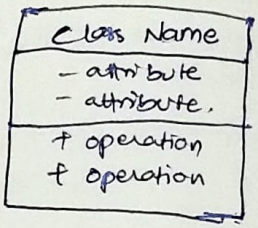
- Initiate and control the flow of activity, while passive classes store data and serve other classes.



Visibility:

- to signify who can access the information contained within a class.
- private - hides information from anything outside the class partition.
- public - allows all other classes to view the marked information
- protected - allows child classes to access information they inherited from a parent class.





Associations

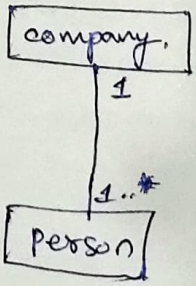
- represent static relationships between classes.
- Roles represent the way the two classes see each other.

NOTE:- it's uncommon to name both the association & the class roles.

Multiplicity (Cardinality)

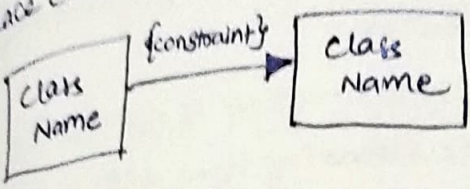
- place multiplicity near the ends of an association.
- these symbols indicate the number of instances of one class linked to one instance of the other class.

1	1
0..1	0/1
*	many
0..*	0/many
1..*	1/many



Constraint

* Place constraint inside { }.

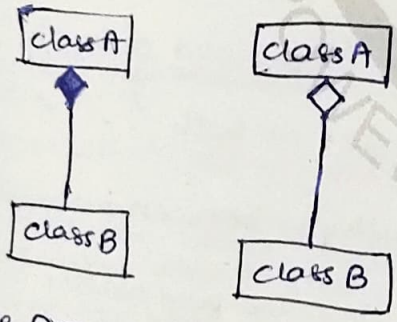


Composition and Aggregation

* Special type of aggregation that denotes a strong ownership between class A, the whole, and class B, its part.
* filled diamond.
* hollow diamond → simple aggregation relationship.

↓

in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other.



Use case Diagram: (courseware)