UNIT I

Overview of Prominent OO Methodologies

System Analysis and Design

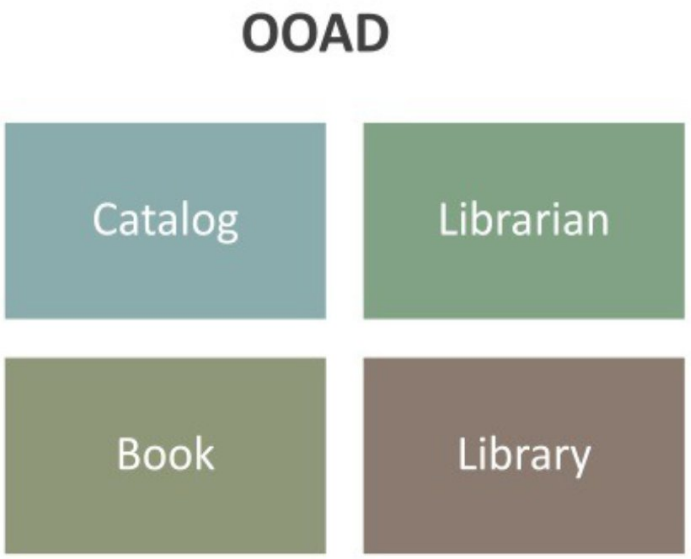
A methodology to develop software systems efficiently.

Key Aspects:

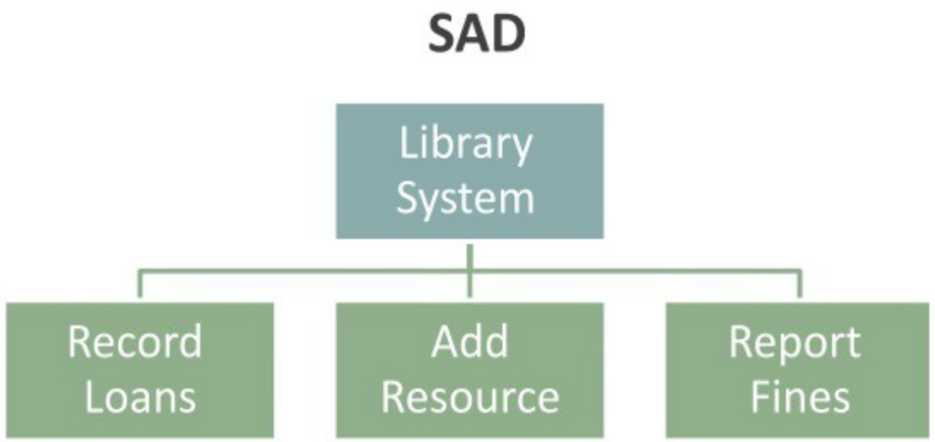
1. **Analysis:** The process of examining information in detail to identify patterns, problems, or key insights. It involves breaking down complex data or ideas to understand them better and describe the core concepts clearly.
2. **Design:** The process of planning and creating a solution to a problem using software. It involves defining the structure, features, and functionality of the software based on requirements and understanding the needs it must address.

System Analysis and Design

Two main approaches: **Structured System Analysis and Design (SSAD)** & **Object-Oriented Analysis and Design (OOAD).**



Partition at the level of **concepts** (objects)



Divide at the **function** level

System Analysis and Design

Structured System Analysis and Design (SSAD)

- A traditional, process-driven methodology focusing on data flow and procedural decomposition.
- **Key Concepts:**
  - Data Flow Diagrams (DFD)
  - Entity-Relationship Diagrams (ERD)
  - Function Decomposition
  - Structured Charts

System Analysis and Design

Object-Oriented Analysis and Design (OOAD)

A modern, object-centric methodology that models systems as interacting objects.

Key Concepts:

- Classes & Objects
- Encapsulation
- Inheritance & Polymorphism
- Use Case Diagrams & UML

System Analysis and Design

Differences between SSAD and OOAD

Feature	SSAD	OOAD
Approach	Process-Oriented	Object-Oriented
Main Focus	Data Flow & Processes	Objects & Interactions
Design Representation	Data Flow Diagrams (DFD), ERD	UML, Class Diagrams
Modularity	Functional Modules	Objects & Classes
Code Reusability	Low	High (via Inheritance)
Risk	High	Low
Flexibility	Less flexible	More adaptable
Best Suited For	Simple, well-defined problems	Complex, scalable systems

Object-Oriented Components

Objects are entities that have attributes, behavior and state.

Classes are sets of objects that share the same attributes, operations, methods, relationships, and semantics.

Properties - Characteristics that define an object.

Simple Property Example: A `Person` object has properties like `name`, `age`, and `height`.

Computed Property Example: A `Rectangle` object may have properties like `width` and `height`, while `area` (computed as `width * height`) is a computed property.

Some properties remain constant, while others can change.

Object-Oriented Components

Attributes - Variables within a class that store object data.

Example: A `Car` object has attributes like `color`, `engine_type`, and `number_of_wheels`.

Only the object is able to change the values of its own attributes.

Attributes vs Properties: Attributes store raw data, while properties may include logic (e.g., computed values like `full_name = first_name + last_name`).

State - The current values held by an object's attributes at a specific time.

Example: A `Car` object may have `speed = 60 km/h` and `fuel = 50%`.

State Changes: State updates based on object interactions.

The set of attribute values defines the state of the object.

Object-Oriented Components

Operations - **Conceptual** descriptions of services an object can provide.

An operation could be:

- question – retrieves data without modifying attribute values.
- command – may modify attribute values.

Example: A `BankAccount` class has an operation `calculateInterest()`, which computes interest based on a formula

Behaviors - Actions that an object can perform.

Example: A `Dog` object can `bark()`, `run()`, and `eat()`.

Defined using methods inside a class.

Object-Oriented Components

Methods - Functions that define an object's behavior.

Example (Python):

```
class Dog:
    def bark(self):
        print('Woof! Woof!')
```

Methods operate on object properties and change the object's state.

Types of Methods

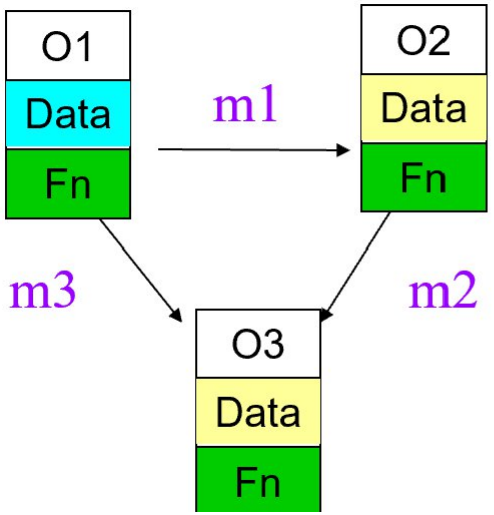
1. Modifier (Mutator): Changes an object's attribute value.
2. Accessor: Returns an object's attribute value.
3. Constructor: Called once when an object is created.
4. Destructor: Called when an object is destroyed.

Object-Oriented Components

Messages - The communication between objects using method calls.

Example: A `User` object calls `login()` on an `Account` object.

Types: Synchronous (direct response) and Asynchronous (delayed response).



{m in Fn}

An Introduction to Object Modeling

- The object-oriented approach is centered around a technique referred to as object modeling.
- Object modeling is a technique for identifying objects within the systems environment, and the relationships between those objects.

Elements of object model

There are four major elements of this object model:

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

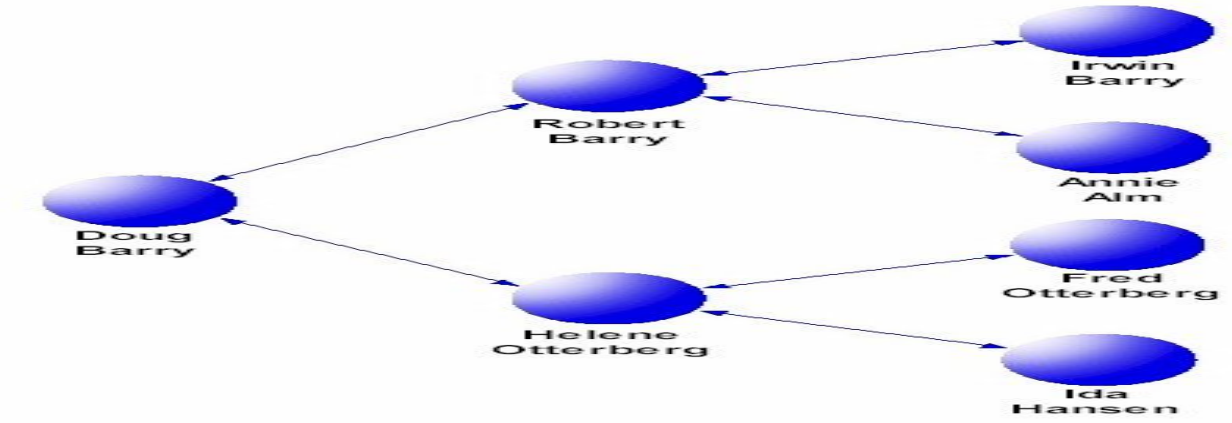
There are another three minor elements of the object model:

- Typing
- Concurrency
- Persistence

An Introduction to Object Modeling

Elements of object model - Abstraction

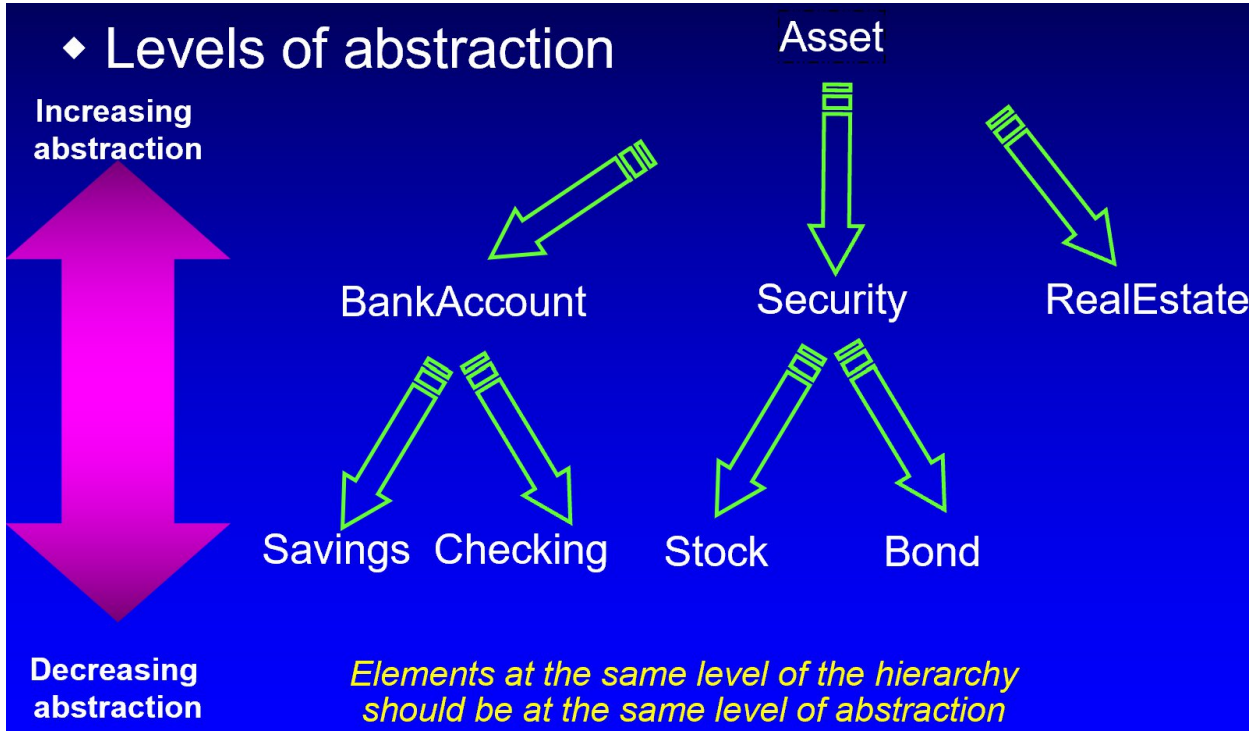
- Abstraction is the process of distilling the data down to its essential.
- But in Object Schema, the abstract data model is implemented as a graph.
- The following figure shows such a graph structure for a genealogical database



An Introduction to Object Modeling

Elements of object model - Abstraction

- Abstraction is the process of modeling only relevant features
  - Hide unnecessary details which are irrelevant for current purpose (and/or user), like eye color
- Reduces complexity and aids understanding
- Done via class, inheritance, association, and aggregation concepts



An Introduction to Object Modeling

Elements of object model - Principle of Abstraction

A process allowing to focus on most important aspects while ignoring less important details.

Abstraction allows us to manage complexity by concentrating on essential aspects making an entity different from others.

An example of an order processing abstraction

An Introduction to Object Modeling

Elements of object model - Encapsulation

Encapsulation is the process of **grouping related data and operations** into a **single unit (class)** while hiding implementation details.

Encapsulation = **Data + Operations** in a **Class**

It separates interface from implementation.

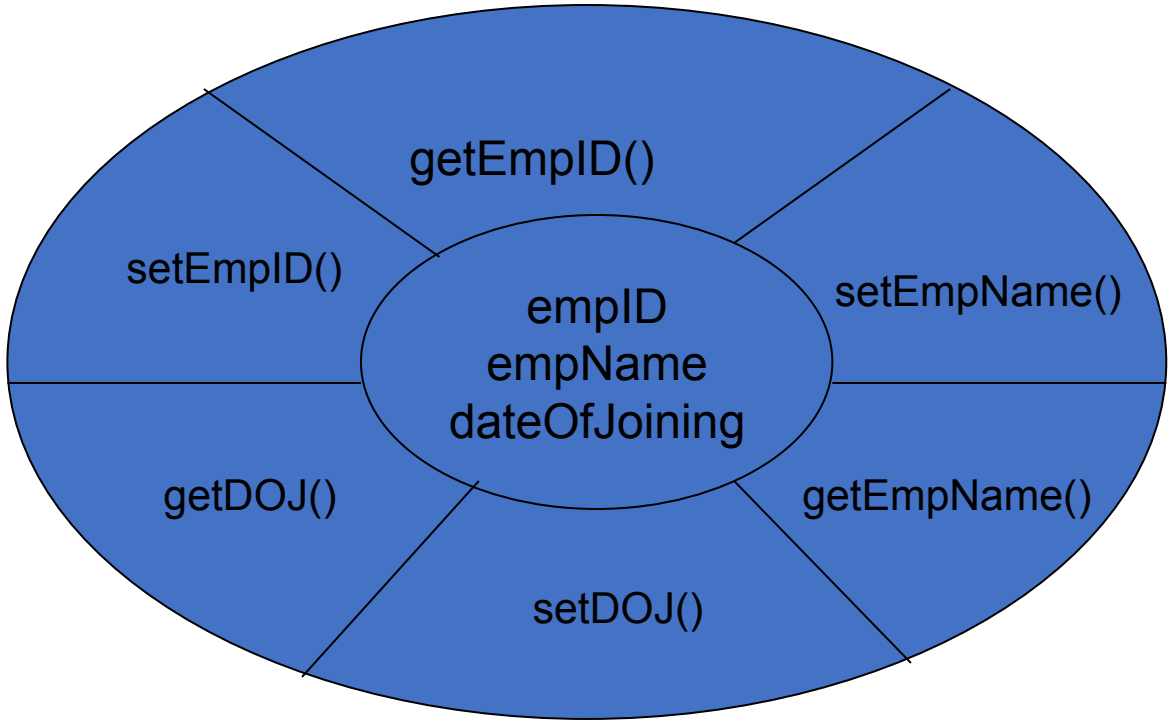
Supports *information hiding* by concealing implementation of the object- users can call a function without knowing internal details (eg., sqrt(x))

Provides controlled access to an object's data

An Introduction to Object Modeling

Elements of object model - Encapsulation

- Encapsulation ensures that **data is accessed and modified only through defined operations**. For example: class is the employee



An Introduction to Object Modeling

Elements of object model - Encapsulation

- Employee class operations:

setEmpID(employeeid:int) = assigns employee to empID

getEmpID():int = returns the value of empID

setEmpName(employeename:string)=assigns employeename to empName

getEmpName():String = returns the name of the employee

setDOJ(doj:date) = assign doj to date of joining

getDOJ:date = returns the date of joining

- For an object e1 of the type Employee,

Set values using “set” methods

Retrieve values using get methods

- Thus, **attributes are hidden** and can only be accessed through **defined methods (interface)**, ensuring **data security and controlled interaction**

An Introduction to Object Modeling

Elements of object model - Encapsulation

It includes three levels of access in Encapsulation :

- Public = All Objects can access it
- Private = Access is limited to members of the same class.
- Protected = Access is limited to members of the same class and its subclasses..

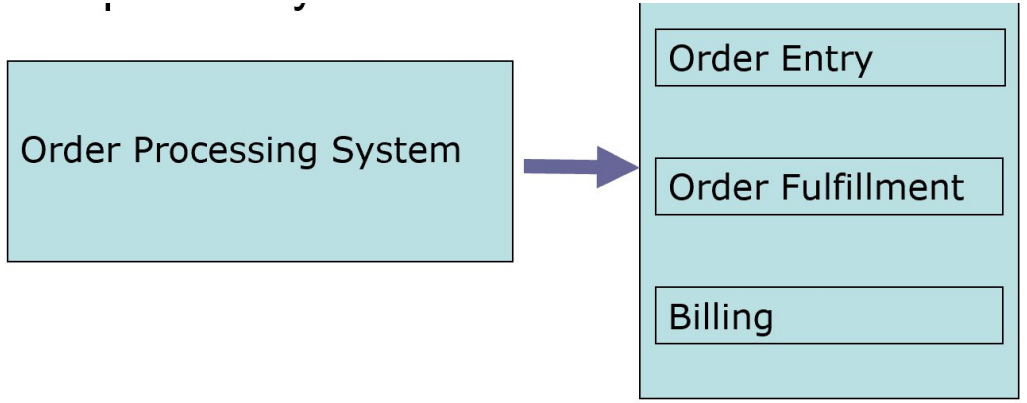
Benefits:

- Improves **modularity** and **security**
- Reduces **complexity** by allowing interaction only through defined operations

An Introduction to Object Modeling

Elements of object model - Modularity

- Modularity breaks up complex systems into small, self-contained pieces that can be managed independently.
- The property of a system where it is divided into **cohesive, loosely coupled modules**.
- Simplifies complex systems by breaking them into smaller, self-contained units that can be managed independently.
- Increases maintainability, scalability, and reusability.



An Introduction to Object Modeling

Elements of object model - Modularity

Coupling - The degree of dependency between modules.

Types:

- Tight Coupling: Modules are highly dependent on each other (harder to modify).
- Loose Coupling: Modules interact with minimal dependencies (preferred for flexibility).

Example: If Module A directly depends on Module B's internal implementation, changing B affects A (tight coupling).

Cohesion - The degree to which elements of a module belong together.

Types:

- High Cohesion: A module has well-defined responsibilities (better for maintainability).
- Low Cohesion: A module does unrelated tasks (harder to manage).

Example: A function that only processes employee salary is highly cohesive, whereas one

An Introduction to Object Modeling

Elements of object model - Hierarchy

- Subclasses *inherit all of the properties and operations* defined for the superclass, and will usually add more
- A hierarchy is an arrangement of items (objects, names, values etc.) in which the items are represented as being “above” , “ below” or “ at the same level as” one another.
- It is a simply an ordered set or an acyclic graph. So, it is called Inheritance.
- **So, Inheritance is a relationships between a super class and its subclasses**

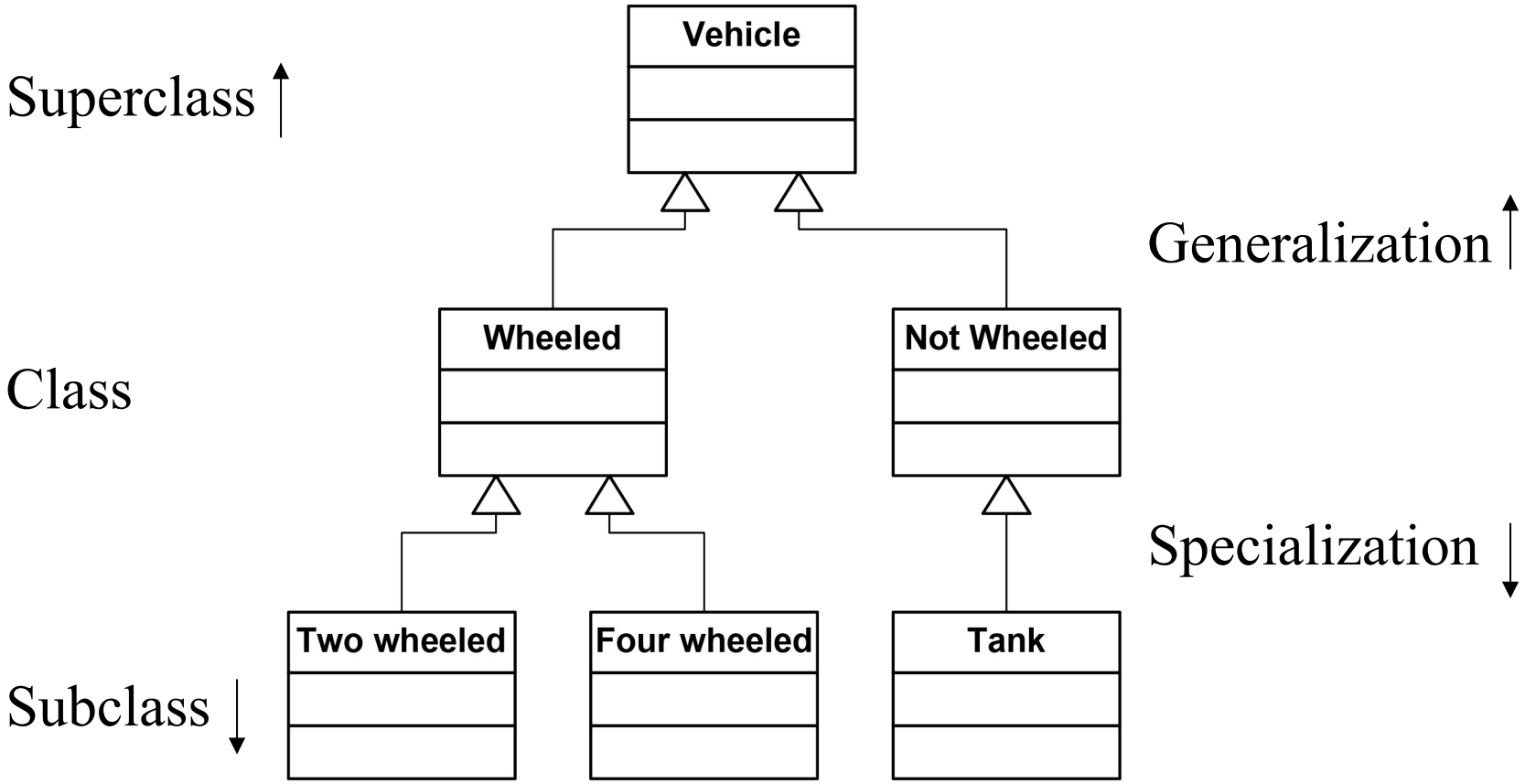
An Introduction to Object Modeling

Elements of object model - Hierarchy

- Inheritance means that all the attributes and operations of an abstract object are available in the specialized object below.
- The triangle in the diagram indicates inheritance.
- The point of the triangle indicates where operations and attributes are inherited from.
- Specialization: The act of defining one class as a refinement of another.
- Subclass: A class defined in terms of a specialization of a superclass using inheritance.
- Superclass: A class serving as a base for inheritance in a class hierarchy
- Inheritance: Automatic duplication of superclass attribute and behavior definitions in subclass.

An Introduction to Object Modeling

Elements of object model - Hierarchy



An Introduction to Object Modeling

Elements of object model – Typing

Typing refers to how a programming language handles data types.

Types:

- Static Typing → Type checked at compile time (e.g., C, Java).
- Dynamic Typing → Type checked at runtime (e.g., Python, JavaScript).

Strong vs. Weak Typing:

- Strongly Typed → Strict type enforcement (e.g., Python).
- Weakly Typed → Flexible type conversion (e.g., JavaScript).

Importance: Proper typing prevents errors and improves efficiency.

An Introduction to Object Modeling

Elements of object model – Concurrency

Concurrency is the ability of a system to execute multiple tasks simultaneously, improving performance, responsiveness, and resource utilization.

Types:

- Multithreading → Multiple threads share the same process.
- Multiprocessing → Multiple processes execute independently.
- Asynchronous Execution → Tasks run in parallel without blocking.

Importance: Improves performance, responsiveness, and resource utilization

An Introduction to Object Modeling

Elements of object model – Persistence

Persistence is the ability to store data beyond the execution of a program, ensuring it remains available and retrievable.

Types:

- File Storage → Saving data in files (e.g., CSV, JSON).
- Database Storage → Using DBMS (e.g., MySQL, MongoDB).
- Object Persistence → Storing objects (e.g., ORM in Django).

Importance: Ensures data is retained, retrievable, and reliable.

CASE Tools

- CASE (Computer-Aided Software Engineering) tools are software applications designed to streamline and enhance various stages of software development.
- They aid in automating tasks such as design, documentation, and even code generation, enabling rapid software production and reducing the overall development effort.
- Characteristics of CASE Tools
    - ❖ Graphic-Oriented Tool
    - ❖ Supports Process Decomposition
    - ❖ Promotes Reusability
    - ❖ Enhances Productivity
    - ❖ Collaboration Support
    - ❖ Code Generation
    - ❖ Rapid Production

Unified Modeling Language (UML)

- Examples of CASE Tools
  - Unified Modeling Language (UML): A tool for visual modeling of systems.
  - Data Modeling Tools: Tools that simplify the creation of database schemas and relationships.
  - Source Code Generation Tools: Applications that can convert models into executable code directly.
- Unified Modeling Language (UML)

  UML is a standard CASE tool that provides a visual representation of object-oriented systems. It is widely used for system design and documentation in software engineering.

Unified Modeling Language (UML)

- UML (Unified Modeling Language)
  - An emerging standard for modeling object-oriented software.
  - Resulted from the convergence of notations from three leading object-oriented methodologies:
    - OMT (James Rumbaugh)
    - OOSE (Ivar Jacobson)
    - Booch (Grady Booch)
- Supported by several CASE tools
  - Rational ROSE
  - ArgoUML
  - TogetherJ

Rumbaugh Methodology (OMT)

- Also known as Object Modeling Technique (OMT)
- It is an object-oriented software development methodology given by James Rumbaugh et.al.
- Used for developing object-oriented systems
- Used in the real world for software modeling and designing
- Main reasons to utilize this modeling approach:
    1. Simulates entities before construction
    2. Reduces complexity through visualization

Rambaugh's Unified Modeling Language (UML)

This methodology describes a method for analysis, design and implementation of a system using object-oriented technique by four phases.

1. Analysis
2. Systems Design
3. Object Design
4. Implementation

ANALYSIS

- An analysis model is a concise, precise abstraction of what the desired system must do, not how it will be done.
- It should not contain any implementation details.
- The objects in the model should be application domain concepts and not the computer implementation concepts.

SYSTEM DESIGN

- The designer makes high level decisions about the overall architecture.
- In system design, the target system is organized as various subsystems based on both the analysis structure and the proposed architecture.

Rambaugh's Unified Modeling Language (UML)

OBJECT DESIGN

- The designer builds a design model based on the analysis model but containing implementation details.
- The focus of object design is the data structures and algorithms needed to implement each cycle.
- The Objects identified in the system design phase are designed.
- Here the implementation of these objects is decided in the form of data structures required and the interrelationships between the objects.

IMPLEMENTATION

- The object classes and relationships developed during object design are finally translated into a particular object oriented programming language, database, or hardware implementation.
- During implementation, it is important to follow good software engineering practice so that the system can remain the traceability, flexibility and extensibility.

Rumbaugh Notation and Models

The OMT consists of three related but different viewpoints and these are described by 3 models:

- Object Model

  Focuses on objects, attributes, and relationships

- Dynamic Model

  Shows interactions using states and events

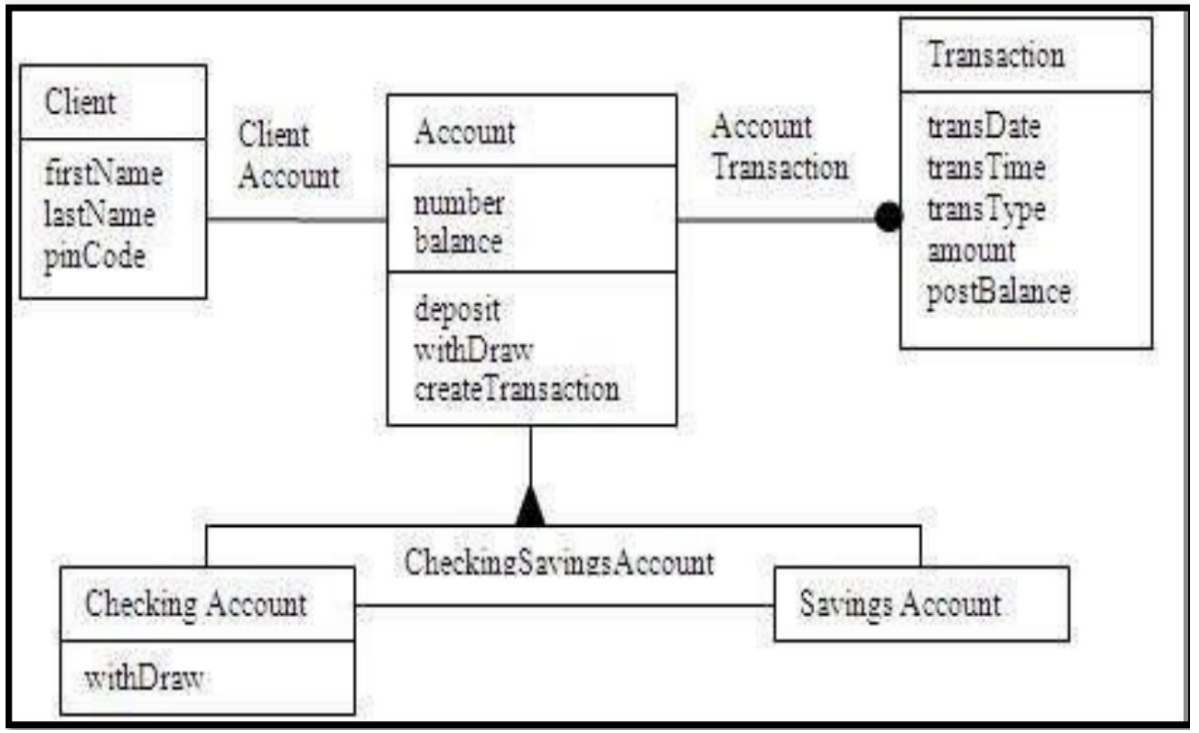
- Functional Model

  Deals with data flow and constraints

Rumbaugh Notation and Models

Object model

- Describes basic structure of objects and their relationships.
- Contains class diagram.
- Class diagram is a graph whose nodes (vertices) are object classes (classes) and whose arcs (edges) are relationships among classes.
- Classes interconnected by association lines .
- Classes- a set of individual objects.

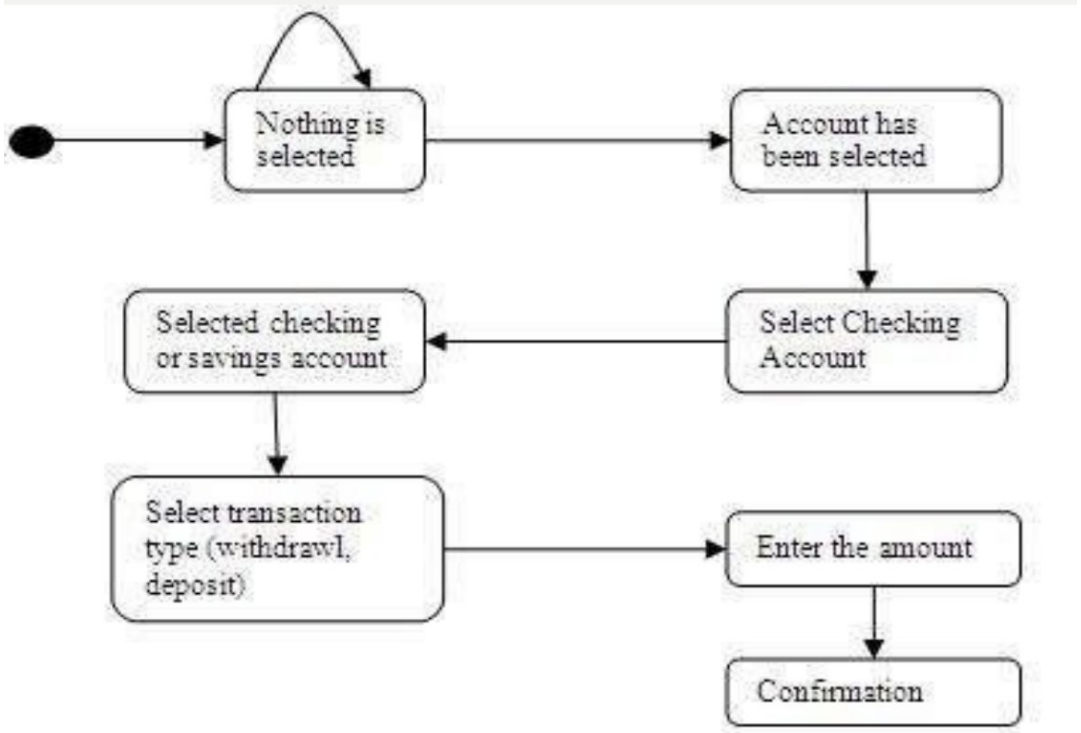


- Association lines- relationship among classes (i.e., objects of one class to objects of another class)

Rumbaugh Notation and Models

Dynamic model

- Describes the aspects of a system that change over time.
- It specifies and implement control aspects of a system.
- Contains state diagram.
- State diagram is a graph whose nodes (vertices) are states and whose arcs (edges) are transitions.



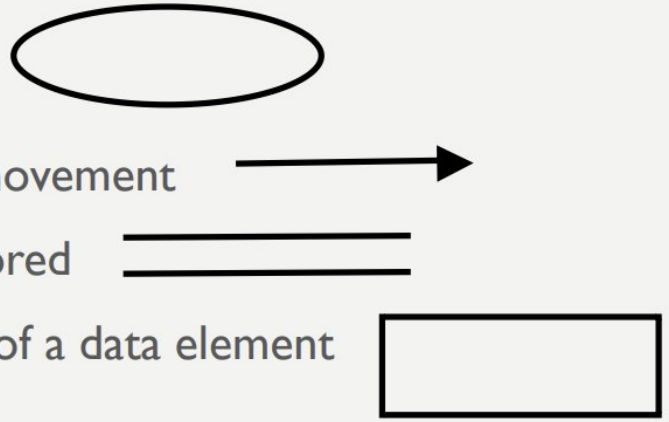
Rumbaugh Notation and Models

Functional model

- Describes the data value transformation within a system.
- It represents how the input data is converted into the required output data.
- Contains data flow diagrams (DFD).
- DFD is a graph whose nodes are process and whose arcs are data-flows

Four primary symbols

- **Process**- any function being performed
- **Data Flow-** Direction of data element movement
- **Data Store –** Location where data is stored
- **External Entity**-Source or Destination of a data element



Rumbaugh Notation and Models

OBJECT vs DYNAMIC vs FUNCTIONAL MODELS

MODEL	REPRESENTS	DIAGRAMS	GRAPH REPRESENTATION	
			VERTICES	EDGES
OBJECT	Static or structural features of a system.	Class diagram	Class	Relationship
DYNAMIC	Aspects of a system that change over time.	State diagram	State	Transition
FUNCTIONAL	Data transformation in a system.	Data Flow Diagram	Process	Data / Dataflow

THE BOOCH METHODOLOGY

It is an object-oriented software development methodology given by Grady Booch et.al.

Diagrams of Booch method

- Class diagrams describe roles and responsibilities of objects
- Object diagrams describe the desired behaviour of the system in terms of scenarios
- State transition diagrams state of a class based on a stimulus
- Module diagrams to map out where each class & object should be declared
- Process diagrams to determine to which processor to allocate a process
- Interaction diagrams describes behaviour of the system in terms of scenarios.

Booch method prescribes:

- Macro Development Process
- Micro Development Process

THE BOOCH METHODOLOGY

- Focuses on object-oriented analysis and design
- The method is cyclical and accounts for incremental improvements that are made in the evolution of a product.
- Macro Development Process

Iterative approach with five activities:

- Conceptualization: Establish requirements taking into account the perspective of the customer
- Analysis: Develop a model by defining object classes, their attributes, methods, and inheritance. Include associations, the dynamic part of a model.
- Design: Develop a structure/architecture where logical and physical details are discussed
- Evolution: As it relates to the implementation
- Maintainance: Maintainance following the delivery of the product

THE BOOCH METHODOLOGY

Micro Development Process

• Each macro process has its own micro development process

Steps:

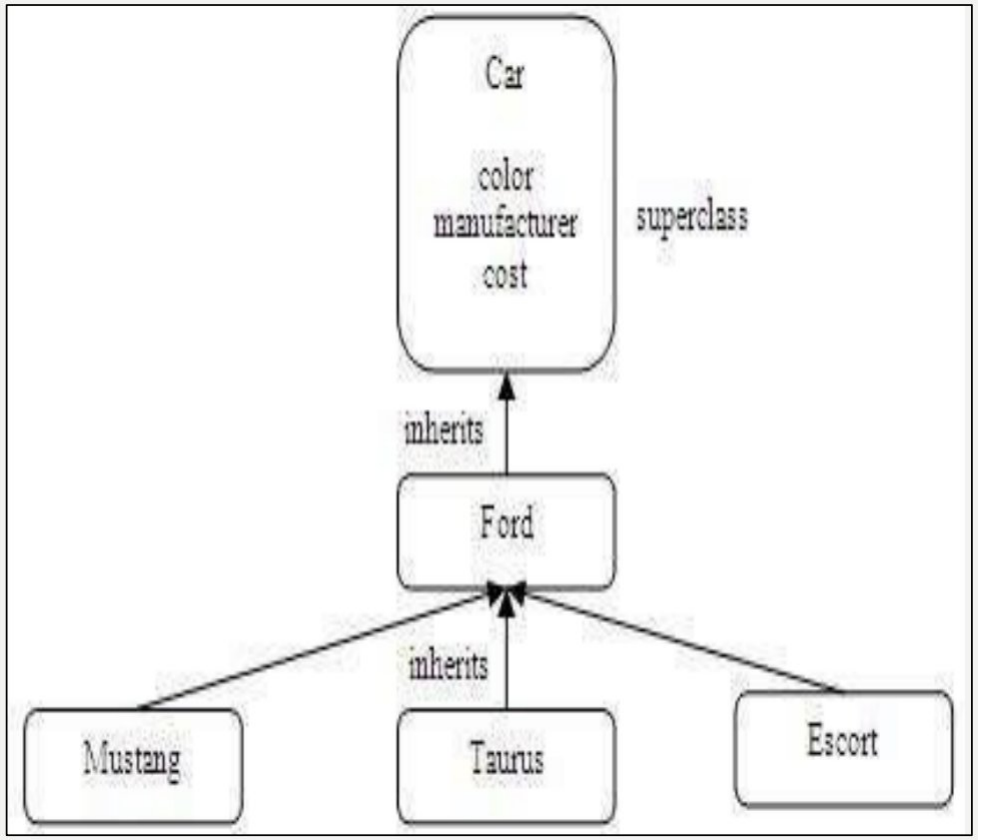
- Identify classes & objects at a given level of abstraction
- Identify class & objects semantics
- Identify class & object relationships
- Identify class & objects interface and implementation

The macro process is the general cycle that follows in Booch's method, the micro process emerges during the development or implementation of new features (classes, behaviors, etc.)

Object modeling using Booch notation.

The arrows represent specialization;

for example, class Taurus is subclass of the class Ford.



THE JACOBSON METHODOLOGIES (OOSE)

Defines interactions between users and the system

- Each Use Case has:
  - Goal for the user
  - Responsibility of the system
- Includes actors, interactions, and constraints

Use Cases

• Understanding system requirements

• Interaction between Users and Systems

The use case description must contain

– How and when the use case begins and ends.

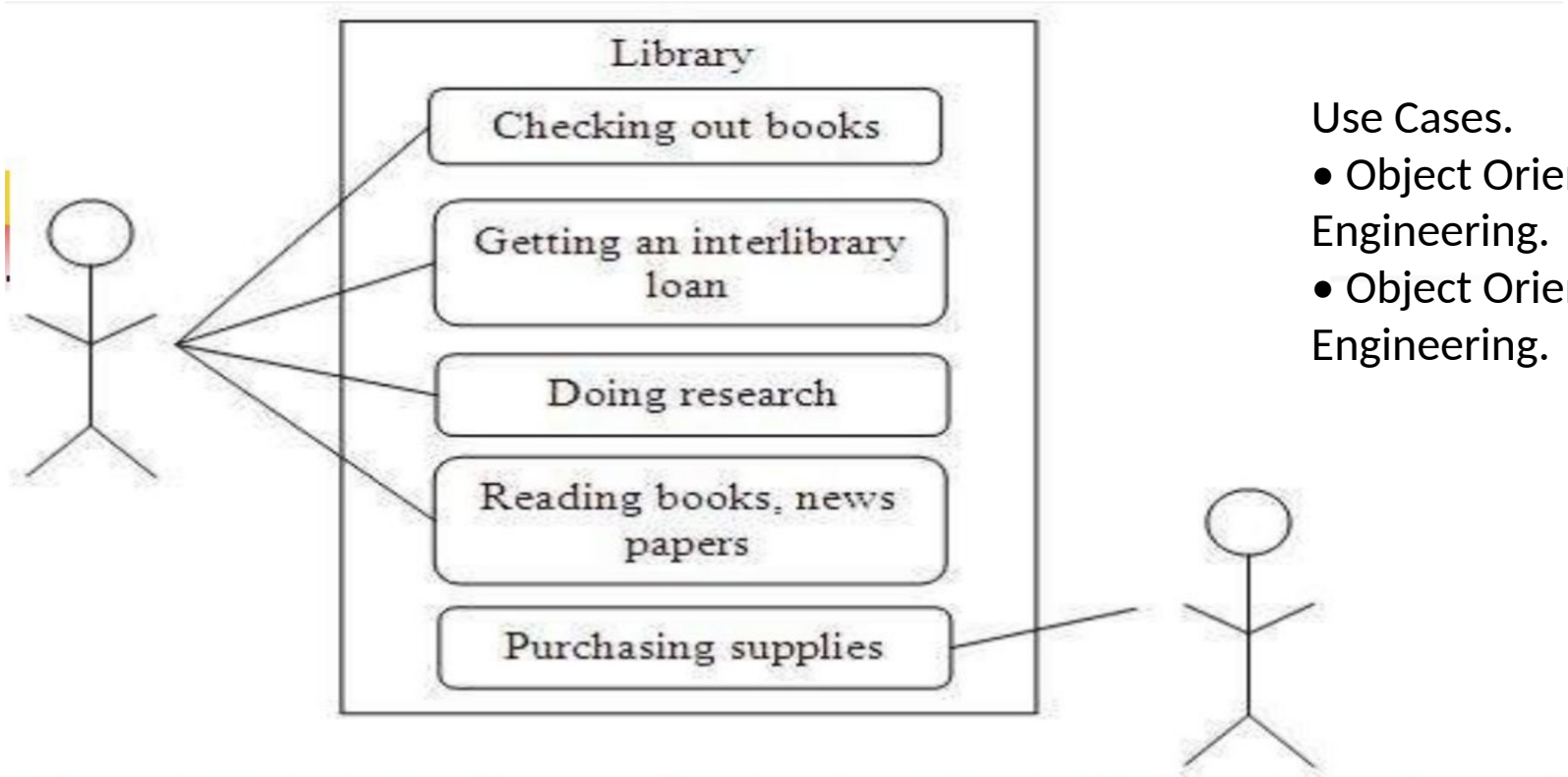
– The Interaction between the use case and its actors, including when the interaction occurs and what is exchanged.

– How and when the use case will need data stored in the system.

– Exception to the flow of events

– How and when concepts of the problem domain are handled.

THE JACOBSON METHODOLOGIES



Use Cases.

- Object Oriented Software Engineering.
- Object Oriented Business Engineering.

Some uses of a library. As you can see, these are external views of the library system from the actor such as a member. The simpler the use case, the more effective it will be. It is unwise to capture all of the details right at the start; you can do that later.

THE JACOBSON METHODOLOGIES

OOSE - Object Oriented Software Engineering.

The Jacobson methodology, also known as Object-Oriented Software Engineering (OOSE) or even Objectory, is a method used to plan, design, and implement object-oriented software.

• Objectory is built around several different models:

– Use case model. The use-case model defines the outside (actors) and inside (use case) of the systems behaviour.

– Domain object model. The objects of the “real” world are mapped into the domain object model.

– Analysis object model. The analysis object model presents how the source code (implementation) should be carried out and written.

– Implementation model. The implementation model represents the implementation of the system.

– Test model. The test model constitutes the test plans, specifications, and reports.

OOBE - Object Oriented Business Engineering

• OOBE is object modeling at the enterprise level.

– Analysis phase

– Design and Implementation phase

– Testing phase

• E.g. Unit testing, integration and system testing.

THE JACOBSON METHODOLOGIES

	Booch Method	*Rumbaugh Method*	*Jacobson Method*
Approach:	Object centered approach.	Object centered approach.	User centered approach.
Phases Covered:	Analysis, design and implementation phases.	Analysis, design and implementation phases.	All phases of life phase cycle.
Strength:	Strong method for producing detailed object oriented design models.	Strong method for producing object model static structure of the system.	Strong method for producing user driven requirements and object oriented analysis model.
Weakness:	Focus entirely on design and not on analysis.	Cannot fully express the requirements.	Do not treat OOP to the same level as other methods.
Uni-directional Relationship:	Uses.	Directed Association.	
Bi-directional Relationship:	Associations.	Uni-directed Associations.	Acquaintance Relationships.
Diagrams used:	Class diagram, state transition diagram, object diagram, timing diagram, Module diagram, process diagram.	Data flow diagrams, state transmission diagram, class/object diagram.	Use case diagram.

The Unified Software Development Process (USDB)

- Developed by Grady Booch, Ivar Jacobson, and James Rumbaugh, the creators of UML.
- A standard and free framework for object-oriented software development.
- Designed to streamline software development and manage project deadlines effectively.
- Closely associated with Rational Unified Process (RUP), later refined by IBM.
- It composes of four phases:
    1. Inception
    2. Elaboration
    3. Conception
    4. Transition

The Unified Software Development Process (USDB)

Characteristics of Unified Software Development Process :

- It is use-case driven - use cases serve as fundamental elements, modeling user interactions and defining entity relationships for system implementation.
- It is architecture-centric - provides a structured architecture that defines system boundaries, ensures model-software alignment, and optimizes resource management.
- It is risk focused - enables early risk identification and resolution in each phase, ensuring smooth software development without delays.
- It is iterative and incremental - software development process which is divided into several phases, where in each phase several stages of work are carried out repeatedly

The Unified Software Development Process

The phases in USDP consist of several iterations and each iteration consists of 5 (five) *workflows* , namely:

1. *Requirement,* aims to identify *user requests (requirements*)

   Identify user needs and express them clearly using **use cases**, defining how users (actors) interact with the system.

2. *Analysis,* the goal is to provide a clearer description and model (based on the description of *the use case*).

   Refine **use cases**, analyze relationships between objects, and identify **static (nouns) and dynamic (verbs) characteristics** of classes.

3. *Design,* in this stage the analysis model will be refined and adapted in a specific platform (*compiler, hardware* , operating system, database and others).

   Convert the analysis model into a **detailed system design**, adapting it to specific platforms (compiler, hardware, OS, database).

4. *Implementation,* this stage transforms the design model into an *executable* program , for example the program will be created in a programming language, compiled , debugged *and* so on.

5. *Test,* the purpose of this stage is to verify and improve *software* performance , such as

The Unified Software Development Process

Inception

- Define vision, business case, and project scope.
- Identify key requirements, risks, and feasibility.
- Align stakeholders and create a **proof of concept prototype**.
- Establish initial **use cases** to model system interactions.

Elaboration

- Refine vision and iteratively develop core architecture.
- Identify most requirements and high-risk elements.
- Expand and validate **use cases** to cover system functionalities.
- Plan resources, costs, and timelines for the next phase.

Construction

- Develop and test solutions iteratively based on **use cases**.
- Maintain architectural integrity and solve critical issues.
- Implement and refine system components from the **prototype**.
- Prepare the system for deployment.

Transition

- Conduct beta testing and deployment.
- Finalize documentation, user manuals, and bug fixes.
- Ensure a smooth transition for end-users based on **use case validations**.

The Unified Software Development Process

Inception is not a requirements phase; rather, it is a feasibility phase, where investigation is done to support a decision to continue or stop. Similarly, elaboration is a phase where the core architecture is iteratively implemented, and high-risk issues are mitigated

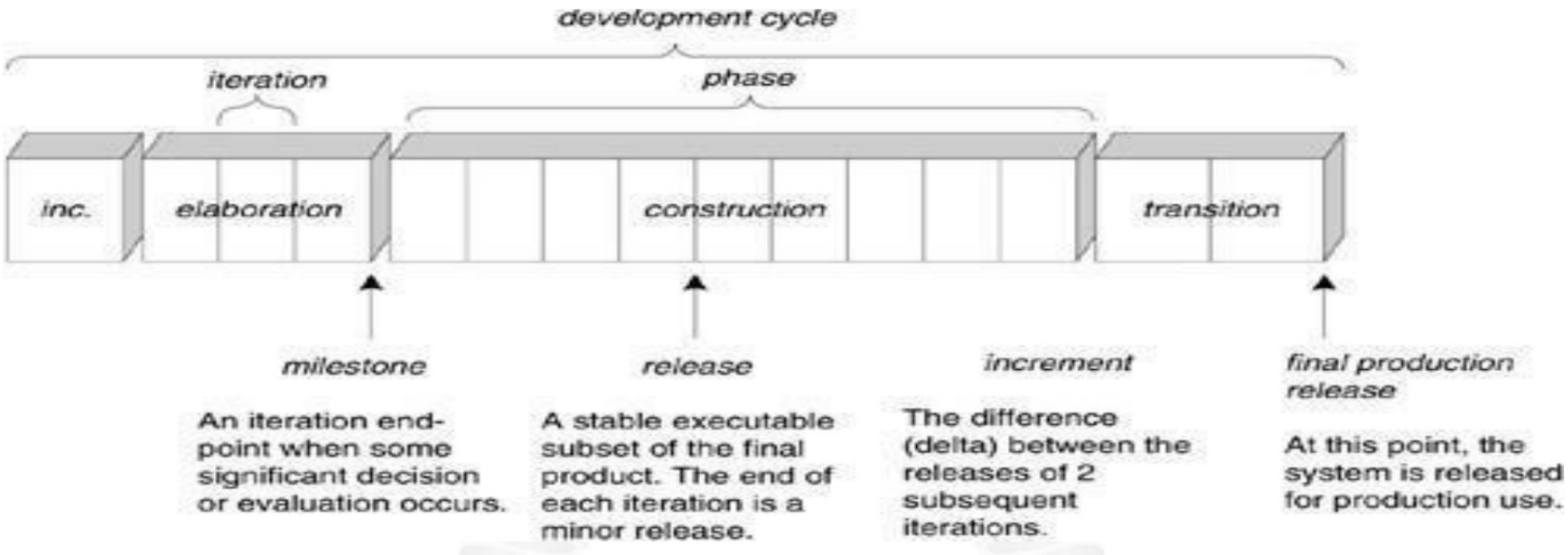


Figure - Schedule-oriented terms in the USDP

The Unified Software Development Process