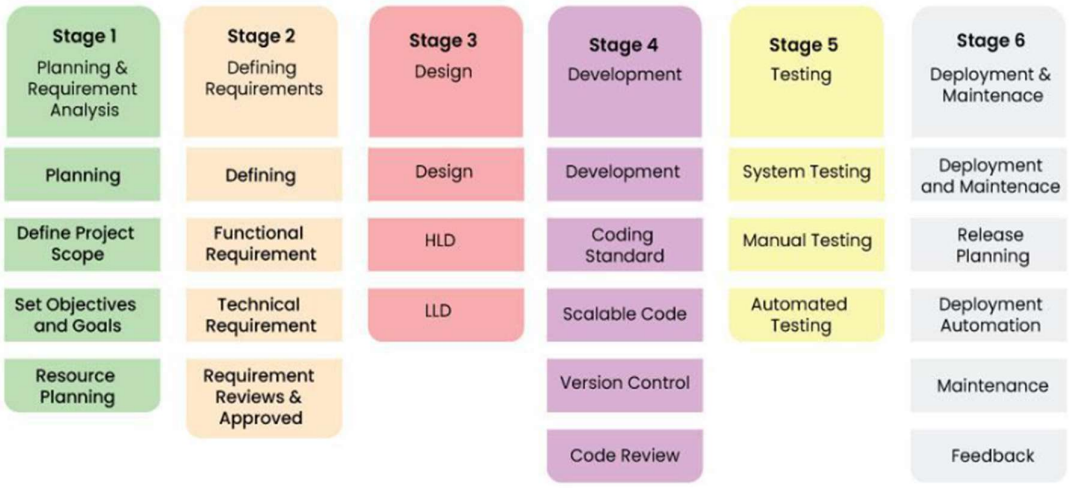
What is SDLC?

SDLC (Software Development Life Cycle) is a step-by-step process used to develop high-quality software that meets user needs. It ensures the software is built efficiently, on time, and within budget.

Stages of SDLC:



1. **Planning & Requirement Analysis**
    - Understand what the customer needs.
    - Gather information and plan the project.
2. **Defining Requirements**
    - Write detailed requirements in a Software Requirement Specification (SRS) document.
    - Get approval from stakeholders.
3. **Designing Architecture**
    - Create design plans based on the SRS.
    - Choose the best design for the software.
4. **Development**
    - Developers write code using programming languages like Java, C++, or Python.
    - Follow design and coding standards.
5. **Testing & Integration**
    - Test the software to find and fix bugs.

- Ensure the product meets the requirements.
- Prepare documentation and provide user training.

6. **Deployment & Maintenance**
    - Release the final product.
    - Fix issues and update software based on user feedback.

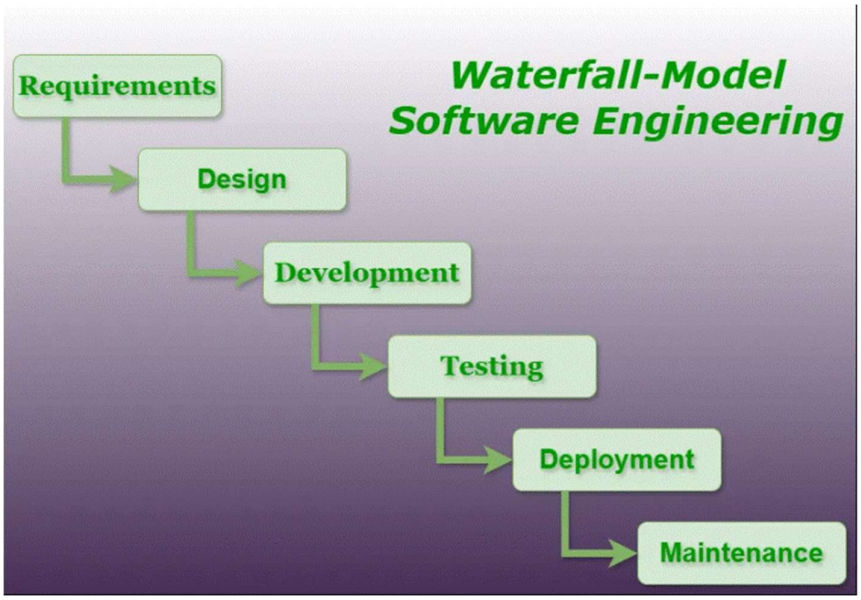
✅ Advantages & Benefits of SDLC:

- **Structured Process:**
  Each step is clearly defined, making the development organized and efficient.
- **Better Quality Software:**
  Testing and validation at every stage ensure fewer bugs and better performance.
- **Improved Project Management:**
  Clear timelines, deliverables, and documentation make it easier to manage the project.
- **Cost-Effective:**
  Early planning reduces the risk of costly rework later.
- **Customer Satisfaction:**
  Ensures the final product meets user expectations.
- **Easy Maintenance:**
  Well-documented software is easier to update and improve.
- **Risk Management:**
  Problems are detected early, reducing overall project risk.

Waterfall Model

The **Waterfall Model** is a linear and step-by-step software development model. Each phase must be completed before moving to the next. It's best for **large, clear, and well-planned projects** with fixed requirements.

🔄 Phases of the Waterfall Model:



1. **Requirement Analysis & Specification**
    - Gather customer requirements.
    - Analyze and document them in the **SRS (Software Requirement Specification)**.
2. **Design**
    - Convert requirements into system design.
    - Two types:
        - 🔷 **High-Level Design (HLD):** Structure of the system.
        - 🔷 **Low-Level Design (LLD):** Details for coding.
3. **Development (Implementation)**
    - Write actual code based on the design.
    - Perform **unit testing** on individual modules.
4. **Testing & Deployment**
    - Combine all modules and test the system.
    - Types of testing:
        - 🔷 **Alpha Testing:** By developers
        - 🔷 **Beta Testing:** By selected users
        - 🔷 **Acceptance Testing:** By customer
    - Deploy the software for actual use.

5. **Maintenance**
    - Fix bugs and make updates after deployment.
    - Types of maintenance:
        - 🔷 **Corrective:** Fix errors
        - 🔷 **Perfective:** Improve features
        - 🔷 **Adaptive:** Modify for new platforms or environments

---

🌟 **Features of Waterfall Model:**

- **Sequential Process:** Follows a strict step-by-step flow.
- **Clear Documentation:** Every phase is documented properly.
- **Emphasis on Testing:** Testing is done after each phase to ensure quality.
- **Good for Fixed Requirements:** Works well when changes are not expected.
- **Easy to Manage:** Simple and easy to understand.

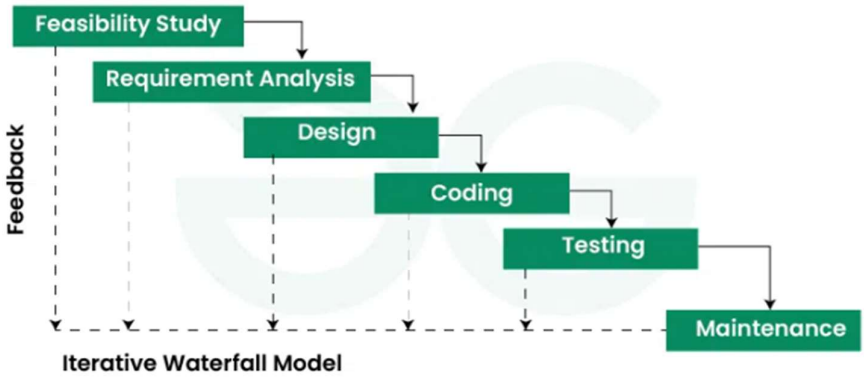
✅ **When to Use the Waterfall Model?**

- When project requirements are **clear and fixed**
- For **large-scale** and **well-defined** IT projects
- When you need **strict planning** and **high-quality** output

Iterative Waterfall Model

The **Iterative Waterfall Model** is an enhanced version of the traditional Waterfall Model. It **follows the same phase-by-phase flow**, but allows **feedback and corrections** at each stage before moving forward. This makes it **more flexible and practical** for real-life projects.

🧩 **Phases of the Iterative Waterfall Model:**



1. **Requirements Gathering**
   → Meet with stakeholders to collect all goals and software needs.
2. **Design**
   → Create a basic design based on the gathered requirements.
3. **Implementation**
   → Start coding and build the system based on the design.
4. **Testing**
   → Test the software to check for errors and see if it meets the requirements.
5. **Deployment**
   → Launch the product for real-world use.
6. **Review and Improvement**
   → Gather user feedback and review performance.
   → Make improvements and repeat the cycle if needed.

📌 **When to Use the Iterative Waterfall Model?**

- When requirements are **clear but may evolve slightly**
- When **new technology** is being used and the team is still learning
- When there's a **high chance of change or risk** in certain project areas
- When a **step-by-step structure** is needed but with room for improvement

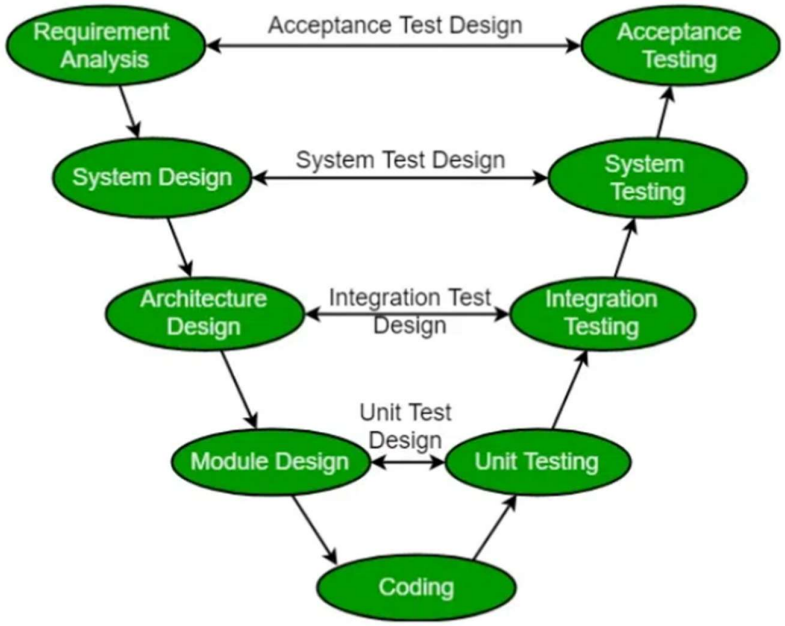
V-Model

The **V-Model** (Verification and Validation Model) is a software development model where **testing is done alongside each development phase**. It is shaped like a "V",

showing how every development stage (left side) has a matching testing stage (right side).

This model is **suitable for projects with clear and fixed requirements**.

🔄 **Phases of V-Model:**



A. Verification Phases *(Left side of the V – Planning & Designing)*

1. **Business Requirement Analysis**
   → Understand what the customer wants.
   → Plan acceptance tests.
2. **System Design**
   → Plan the full structure and behavior of the software.
   → Decide how the software will work.
3. **Architectural Design (High-Level Design)**
   → Break the system into modules and define how they interact.
   → Plan integration tests.
4. **Module Design (Low-Level Design)**
   → Design the internal logic of each module.
   → Plan unit tests.
5. **Coding**
   → Write actual code based on the designs.
   → Follow coding standards and do code reviews.

B. Validation Phases *(Right side of the V – Testing)*

1. **Unit Testing**
   → Test each module individually.
   → Based on the module design.
2. **Integration Testing**
   → Test how modules work together.
   → Based on the architectural design.
3. **System Testing**
   → Test the whole software system.
   → Ensure it meets both functional and non-functional requirements.
4. **User Acceptance Testing (UAT)**
   → Test in a real-like user environment.
   → Confirm the software meets business needs.

🌟 **Key Features of V-Model:**

- **Early Testing:** Testing starts in the planning/design phase.
- **Clear Structure:** Each development step has a matching test step.
- **Easy to Manage:** Simple and organized with strict documentation.
- **Best for Fixed Requirements:** Works well when the scope is clear and won't change.
- **Improves Quality:** Bugs are caught early due to planned testing.

Spiral Model

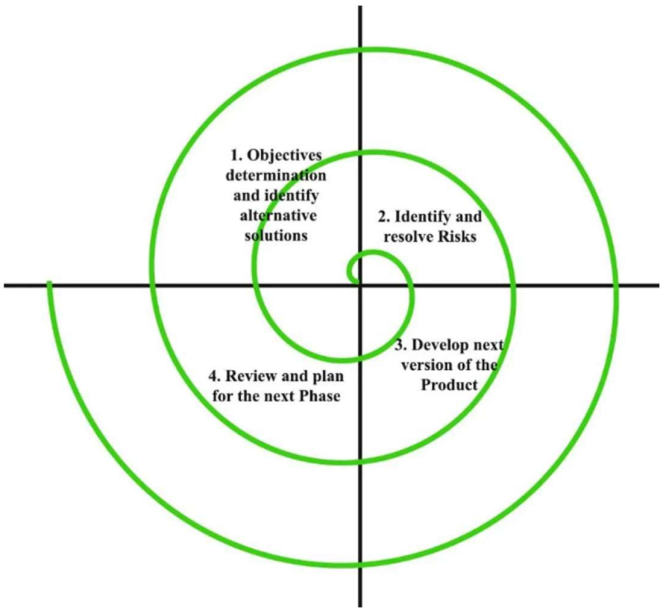
The **Spiral Model** is a **combination of the Waterfall and Iterative models**, focusing heavily on **risk management**. It was proposed by **Barry Boehm** and is best suited for **large, high-risk projects**.

It looks like a **spiral with multiple loops**, and each loop represents one full development cycle (planning → design → development → testing → review).

📌 **Key Highlights:**

- It is **risk-driven** – risks are identified and handled in each cycle.
- The **number of cycles (loops)** is not fixed and can vary based on project needs.
- Each cycle improves the product step-by-step.

🔁 **Phases of the Spiral Model (in 4 Quadrants)**



1. Objectives Definition (Planning Phase)

→ Gather requirements and define goals for this development cycle.

→ List functional and non-functional requirements.

→ Propose multiple possible solutions.

2. Risk Analysis and Prototyping

→ Identify and analyze all possible risks.

→ Choose the best solution and resolve risks.

→ Build a prototype of the chosen solution.

3. Development and Testing

→ Develop the selected version of the product.

→ Test the software for issues and ensure quality.

4. Customer Evaluation and Planning

→ Customer reviews the progress and gives feedback.

→ Plan the next loop based on this feedback.

→ Start a new cycle with improved understanding.

🌟 **Features of the Spiral Model:**

- **Focus on Risk Handling:** Unique among SDLC models for its risk analysis phase.

- **Iterative Approach:** Allows repeated improvements based on feedback.
- **Customer Involvement:** Clients are involved in each loop, improving product quality.
- **Flexible and Customizable:** Phases and iterations vary as per project needs.

Prototyping Model

- Used when **requirements are not clearly known** at the start.
- A **prototype** (rough version) is built first, shown to the user, improved based on feedback, and repeated until the user is satisfied.
- The final product is built based on the approved prototype.

🔷 Phases of Prototyping Model

1. **Requirement Gathering**
    - Understand what the user needs through interviews.
2. **Quick Design**
    - Make a rough design with basic features.
3. **Build Prototype**
    - Create a working version with limited features.
4. **User Evaluation**
    - Show prototype to the user and get feedback.
5. **Refine Prototype**
    - Improve based on feedback and repeat until approved.
6. **Final Product Development & Maintenance**
    - Develop the final system and maintain it regularly.

🔷 Types of Prototyping

1. **Rapid Throwaway Prototyping**
    - Quickly build, get feedback, and discard. Helps avoid major design mistakes.
2. **Evolutionary Prototyping**
    - Keep improving the same prototype until it's accepted.
3. **Incremental Prototyping**

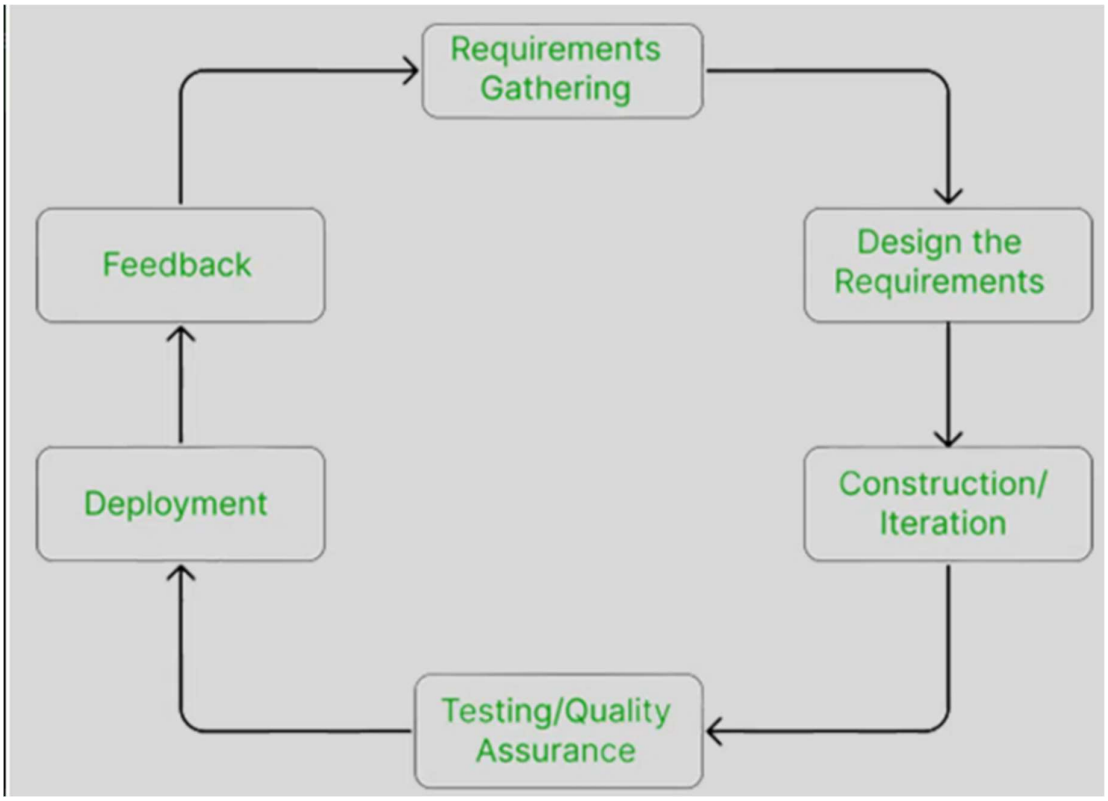
- Build small parts (modules) separately and combine at the end.

4. **Extreme Prototyping** (For Web Apps)
    - Step 1: Static HTML pages
    - Step 2: Simulate functionality
    - Step 3: Connect real services

🔷 Agile Model

- The **Agile Model** focuses on **fast delivery**, **flexibility**, and **continuous improvement**.
- It supports **quick adaptation to changes** and encourages **regular customer feedback**.
- Combines **iterative** (repeating) and **incremental** (adding step-by-step) development.

🔷 Phases of Agile Model



1. **Requirement Gathering**
    - Understand what the user wants.
    - Estimate time, cost, and check if the project is feasible.

2. **Design the Requirements**
    - Create rough designs (wireframes, UML diagrams).
    - Make prototypes to get early feedback.
3. **Construction / Iteration**
    - Develop the software in small cycles (1–4 weeks).
    - Each cycle delivers a working version with new features.
4. **Testing / Quality Assurance**
    - Perform:
        - **Unit Testing** – test individual parts.
        - **Integration Testing** – test combined parts.
        - **System Testing** – test the full system.
5. **Deployment**
    - Release the working software to users.
    - Deployment is continuous and frequent.
6. **Feedback**
    - Collect user feedback.
    - Fix bugs and improve features.
    - Plan changes for the next cycle.

UML

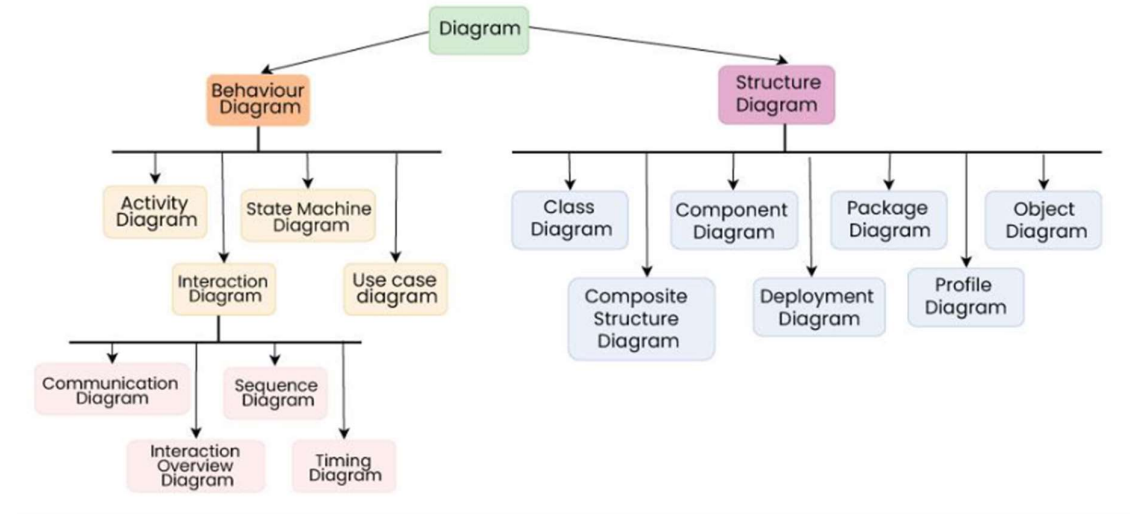
- **UML (Unified Modeling Language)** is a standard way to **visually design, document, and understand** software systems.
- It shows how a system behaves and how its parts are structured.
- It's useful for **software engineers, system architects, and business people** to plan and communicate clearly.

🔷 Why Do We Need UML?

- To **visualize complex systems** easily.
- Helps teams **collaborate better**, especially when different roles are involved (e.g., developers and business clients).
- Saves time by **clarifying designs early**, reducing misunderstandings.

- **Non-coders** can understand system workflows using diagrams.

🔷 Types of UML Diagrams

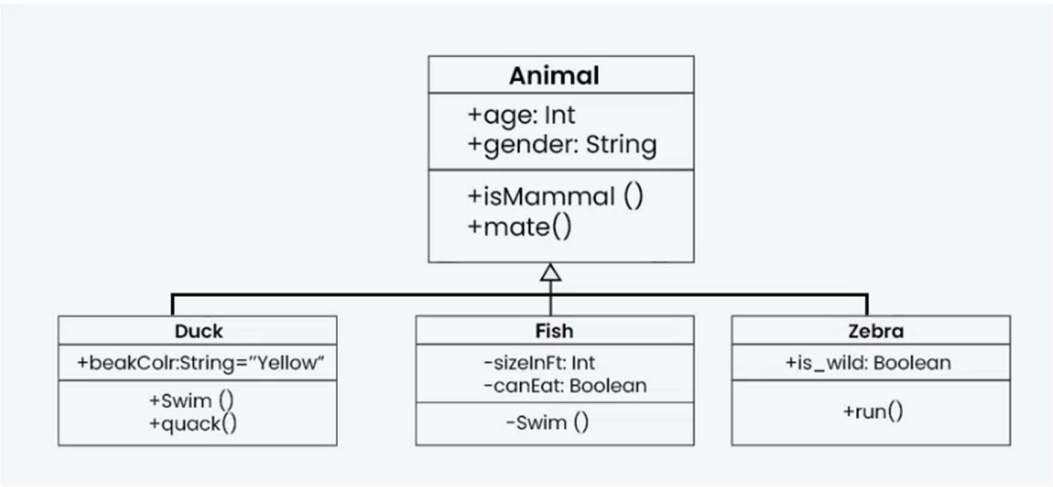


UML diagrams are divided into two main categories:

🧱 1. Structural Diagrams

These show the **static structure** of the system (what it *has*).

1. **Class Diagram**



   Shows classes, attributes, methods, and relationships between classes.

2. **Object Diagram**
   Represents instances (objects) of classes at a specific point in time.

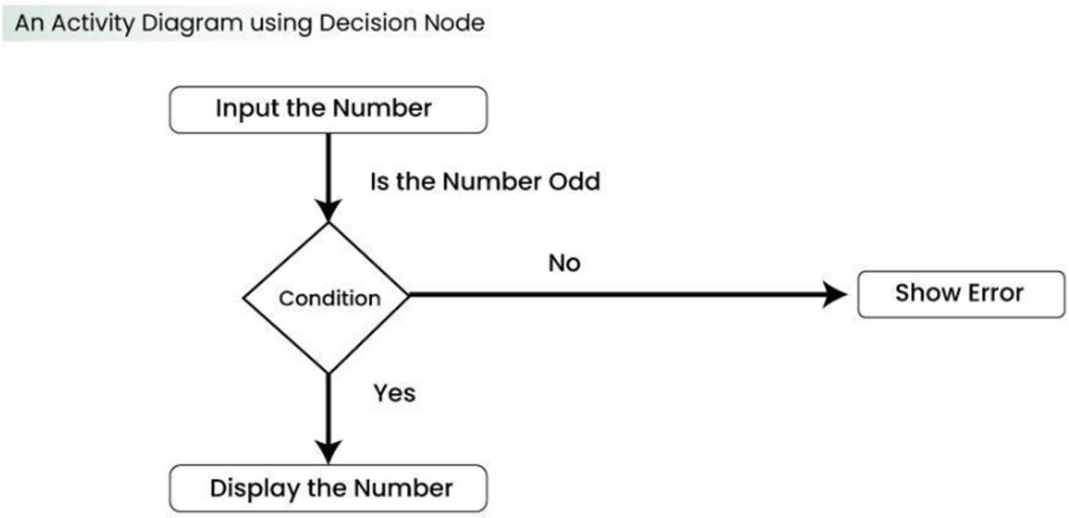
3. **Component Diagram**
   Shows how software components (modules) are organized and interact.

4. **Deployment Diagram**
   Displays hardware components and the software running on them.
5. **Package Diagram**
   Organizes and shows dependencies between different packages.
6. **Composite Structure Diagram**
   Represents the internal structure of a class and its interaction points.

🔁 2. Behavioral Diagrams

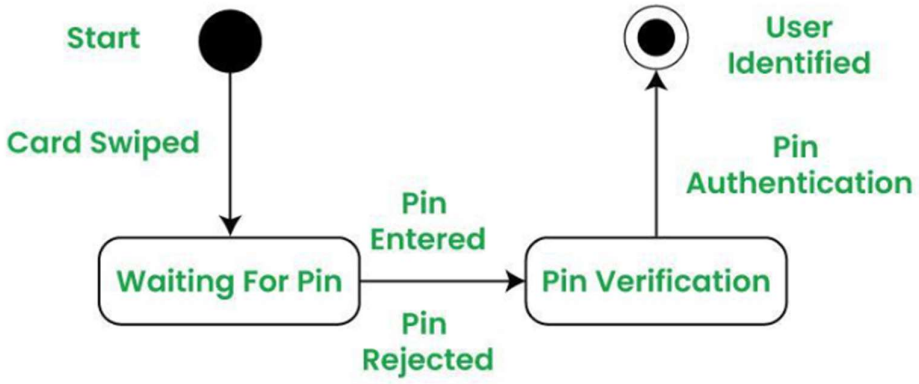
These show the **dynamic behavior** of the system (how it *works* or *responds*).

1. **Use Case Diagram**
   Shows system functionality and interactions with external users (actors).
2. **Activity Diagram**
   Represents workflows or step-by-step processes within the system.



3. **State Machine Diagram**

A State Machine Diagram for user verification



Shows different states of an object and how it transitions between them.

4. **Sequence Diagram**
   Depicts how objects communicate with each other in a time-based sequence.
5. **Communication Diagram**
   Focuses on object interactions and the messages they exchange.
6. **Timing Diagram**
   Shows changes in object behavior over time with timing constraints.
7. **Interaction Overview Diagram**
   Gives a high-level overview of interactions, combining activity and sequence diagrams.