Software Engineering

- Vineela

Sessions 2, 3, 4 & 5

- Introduction to software engineering
  - o Software Process
  - o Software Process Model
  - o Software Product
- Importance of Software engineering
- Software Development Life Cycles
- Requirements Engineering
  - o Types of Requirements
  - o Steps involved in Requirements Engineering
  - o Requirement Analysis Modelling
- Design and Architectural Engineering
  - o Characteristics of Good Design
  - o Function Oriented vs Object Oriented System
  - o Modularity, Cohesion, Coupling, Layering
  - o Design Models
  - o UML
- Coding
  - o Programming Principles
  - o Coding Conventions
- Object Oriented Analysis and Design

Introduction to Software Engineering

- Software Engineering is the branch of computer science that applies engineering principles and deals with the design, development, testing and maintenance of software applications

- It’s the science and art of building good software on time, within budget and with high quality.

What is the goal of Software Engineering?

- To produce high-quality, dependable and effective software while completing the project on time and within a given budget.

Software Process

- A software process (also called a software development process) is a structured set of activities required to develop a software system.
- It defines how software is built, from the idea stage to final deployment and maintenance.

Think of it as a **roadmap that guides software engineers** through the planning, development, and delivery of software ensuring that teams work methodically, efficiently and with consistent quality.

Key Activities in a Software Process

- Requirements Gathering – Understand what the user needs.
- Design – Plan the architecture and components of the system.
- Implementation (Coding) – Write the actual code.
- Testing – Verify that the software works as intended.
- Deployment – Release the software to users.
- Maintenance – Fix bugs and add enhancements over time.

General Software Process Flow

[Requirement Analysis] → [System Design] → [Coding] → [Testing] → [Deployment] →[Maintenance]

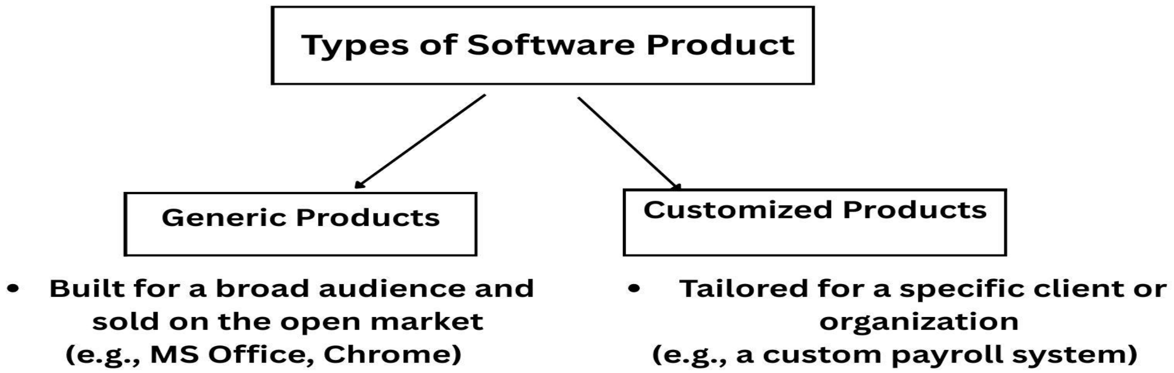
Common Software Process Model

Model	Description	Used For
Waterfall	Linear and sequential. Each phase must be completed before the next begins.	Small, well-defined projects **Ex:** Supermarket Website
Agile	Iterative (provides feedback paths from every phase to its preceding phases) and flexible. Emphasizes collaboration and adaptability.	Dynamic projects with changing requirements
Spiral	Combines iterative (Feedback is taken, so changes at each stage) development with risk analysis.	Large, high-risk projects
V-Model	Each development phase has a corresponding testing phase.	Projects requiring strong testing **Ex**: Medical Application
Prototyping	Build prototypes to gather feedback early.	Projects with unclear requirements
Incremental	Builds software in small, manageable pieces (increments)	Projects needing early partial delivery

Software Product

Software Product is any Software or any system which has been developed, tested and maintained for the specific purpose to solve the problem for the benefit of a user base

Ex - users hold online video conferences and meetings (think Zoom) or a platform that helps businesses manage employee expenses.



Types of Software Product

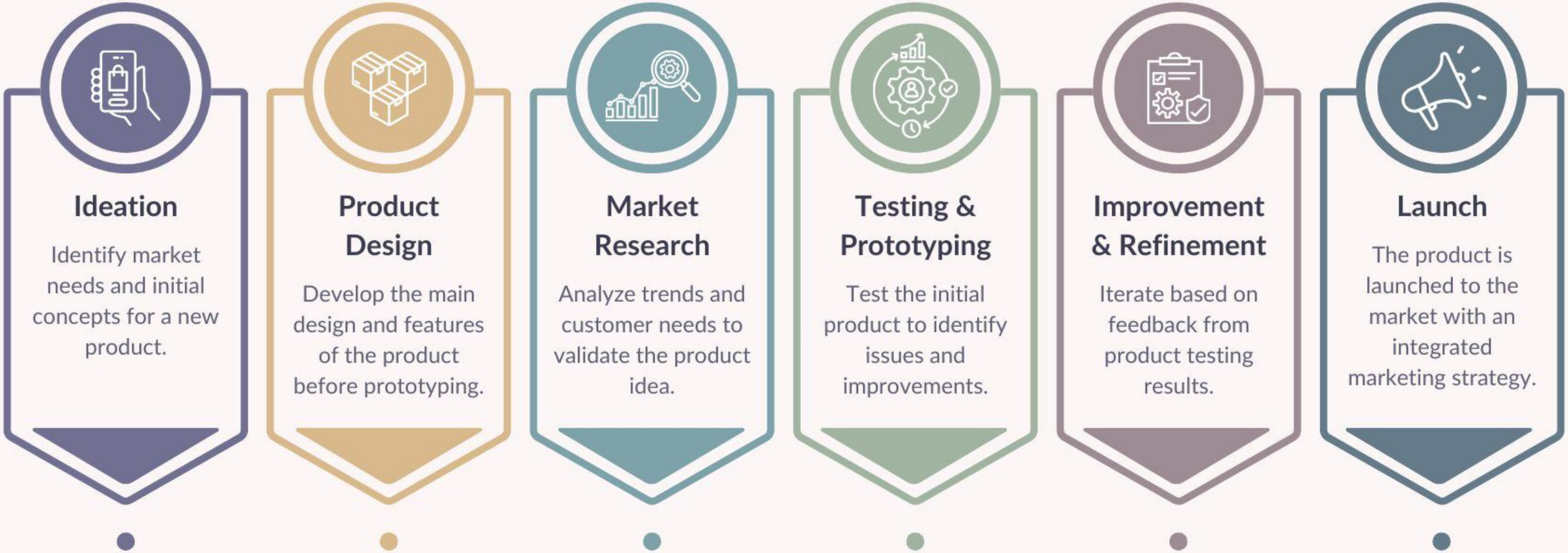
Generic Products

- Built for a broad audience and sold on the open market (e.g., MS Office, Chrome)

Customized Products

- Tailored for a specific client or organization (e.g., a custom payroll system)

PRODUCT DEVELOPMENT ROADMAP



Ideation

Identify market needs and initial concepts for a new product.

Product Design

Develop the main design and features of the product before prototyping.

Market Research

Analyze trends and customer needs to validate the product idea.

Testing & Prototyping

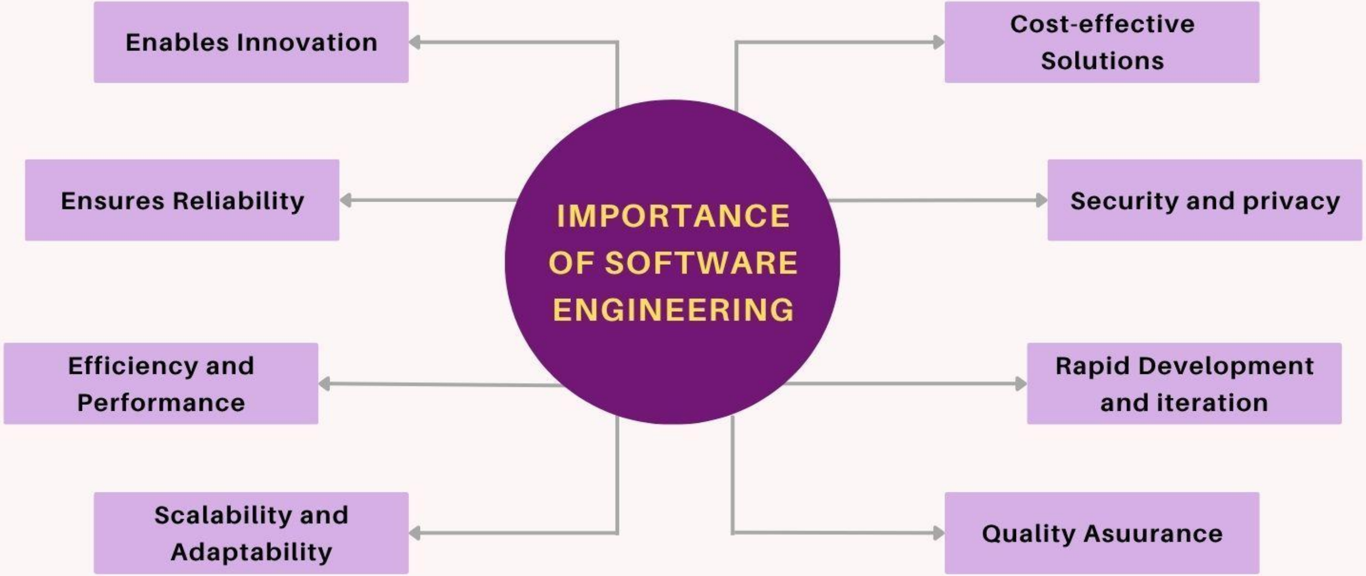
Test the initial product to identify issues and improvements.

Improvement & Refinement

Iterate based on feedback from product testing results.

Launch

The product is launched to the market with an integrated marketing strategy.



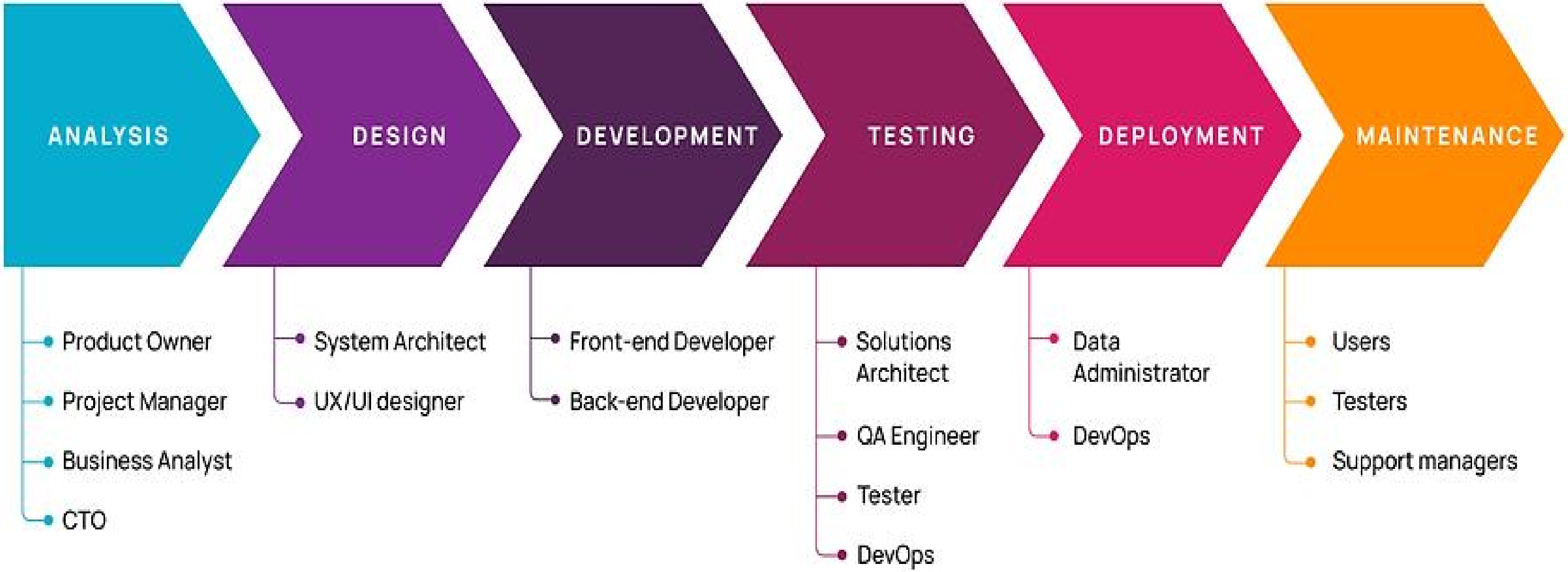
IMPORTANCE OF SOFTWARE ENGINEERING

- Enables Innovation
- Ensures Reliability
- Efficiency and Performance
- Scalability and Adaptability
- Cost-effective Solutions
- Security and privacy
- Rapid Development and iteration
- Quality Asuurance

CHARACTERISTICS OF SOFTWARE ENGINEERING

Efficiency	should not make wasteful use of system resources
Maintainability	possible to evolve the software to meet the changing requirements
Dependability	ought to not cause any physical injury in case of failure
In time	developed well in time
Within Budget	should be within the budgetary limit
Functionality	should perform all the functions
Adaptability	ability to adapt to a reasonable extent
Security and Privacy	helping to protect data and systems from cyber threats

6 Phases of the Software Development Life Cycle



ANALYSIS

- Product Owner
- Project Manager
- Business Analyst
- CTO

DESIGN

- System Architect
- UX/UI designer

DEVELOPMENT

- Front-end Developer
- Back-end Developer

TESTING

- Solutions Architect
- QA Engineer
- Tester
- DevOps

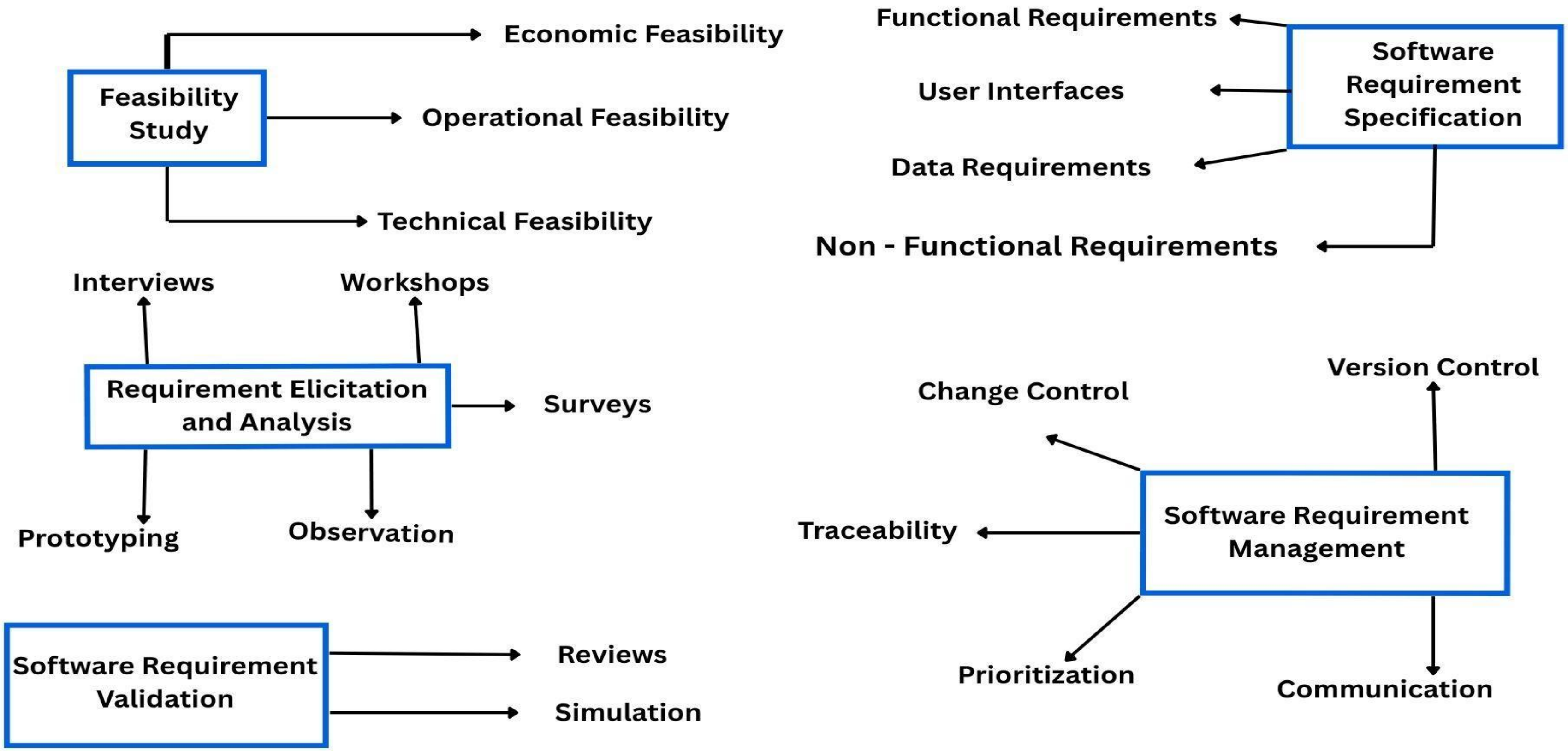
DEPLOYMENT

- Data Administrator
- DevOps

MAINTENANCE

- Users
- Testers
- Support managers

Requirements Engineering



Feasibility Study

- Economic Feasibility
- Operational Feasibility
- Technical Feasibility

Software Requirement Specification

- Functional Requirements
- User Interfaces
- Data Requirements
- Non - Functional Requirements

Requirement Elicitation and Analysis

- Interviews
- Workshops
- Surveys
- Prototyping
- Observation

Software Requirement Management

- Version Control
- Change Control
- Traceability
- Prioritization
- Communication

Software Requirement Validation

- Reviews
- Simulation

Requirements Engineering

1) **Feasibility Study**: This is the **initial phase** of the requirement engineering process. It involves evaluating the feasibility of the proposed software project.
    - Technical Feasibility
    - Economic Feasibility
    - Operational Feasibility
1) **Requirement Elicitation and Analysis** : The goal is to understand the needs of the system and translate them into specific software requirements by gathering and analyzing requirements from stakeholders, including users, customers, and domain experts
    - Interviews, Surveys,Workshops, Observation, Prototyping
1) **Software Requirement Specification** : The Software Requirement Specification (SRS) document is created, which serves as a blueprint for the development process.

    SRS includes the following components:

    - Functional Requirements: Descriptions of what the system should do in terms of specific functions and features.
    - Non-functional Requirements: Constraints and qualities the system must possess (e.g., performance, security).
    - User Interfaces: Descriptions of how users will interact with the system.
    - Data Requirements: Details about the data the system will store, process, and manage.

Requirements Engineering

4) **Software Requirement Validation -** The aim is to identify and rectify any errors or misunderstandings before development begins by validating the documented requirements to represent the stakeholders' needs and expectations

- Reviews, Prototyping, Simulation

5) **Software Requirement Management -** This **i**nvolves maintaining and tracking changes to requirements throughout the software development lifecycle

- Version Control, Change Control,Traceability, Prioritization, Communication

Requirement Analysis Modelling

- Requirement Analysis Modeling is the process of **visually and structurally representing software requirements** to better understand, analyze, and communicate them.
- It bridges the gap between what stakeholders want and how developers interpret those needs.

Why Use Modeling in Requirement Analysis?

- Clarifies complex requirements
- Identifies inconsistencies or gaps early
- Improves communication between stakeholders and developers
- Supports validation and verification of requirement

Common Modeling Techniques

Technique	Description
Use Case Diagrams (UML)	Show interactions between users (actors) and the system
Data Flow Diagrams	Visualize how data moves through the system
Entity-Relationship Diagrams	Model data entities and their relationships
Class Diagrams (UML)	Represent system structure using classes and relationships
Business Process Modeling	Graphically represent business processes using standardized symbols.
Flowcharts	Show step-by-step logic or control flow
State Diagrams	Describe how a system behaves in response to events

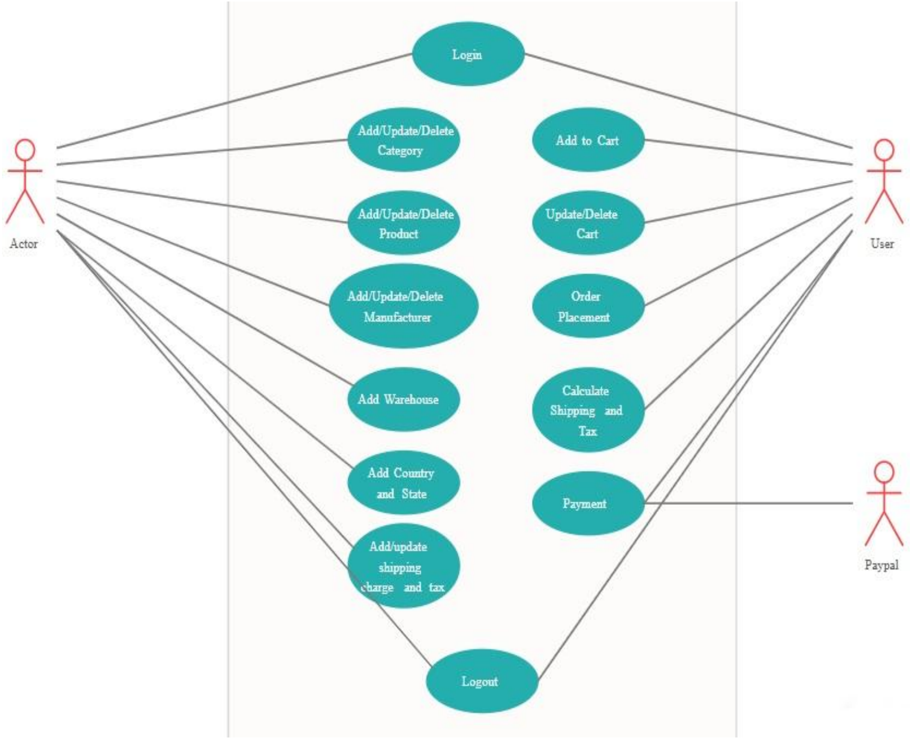
Example Scenario

Imagine you're building an online food delivery app:

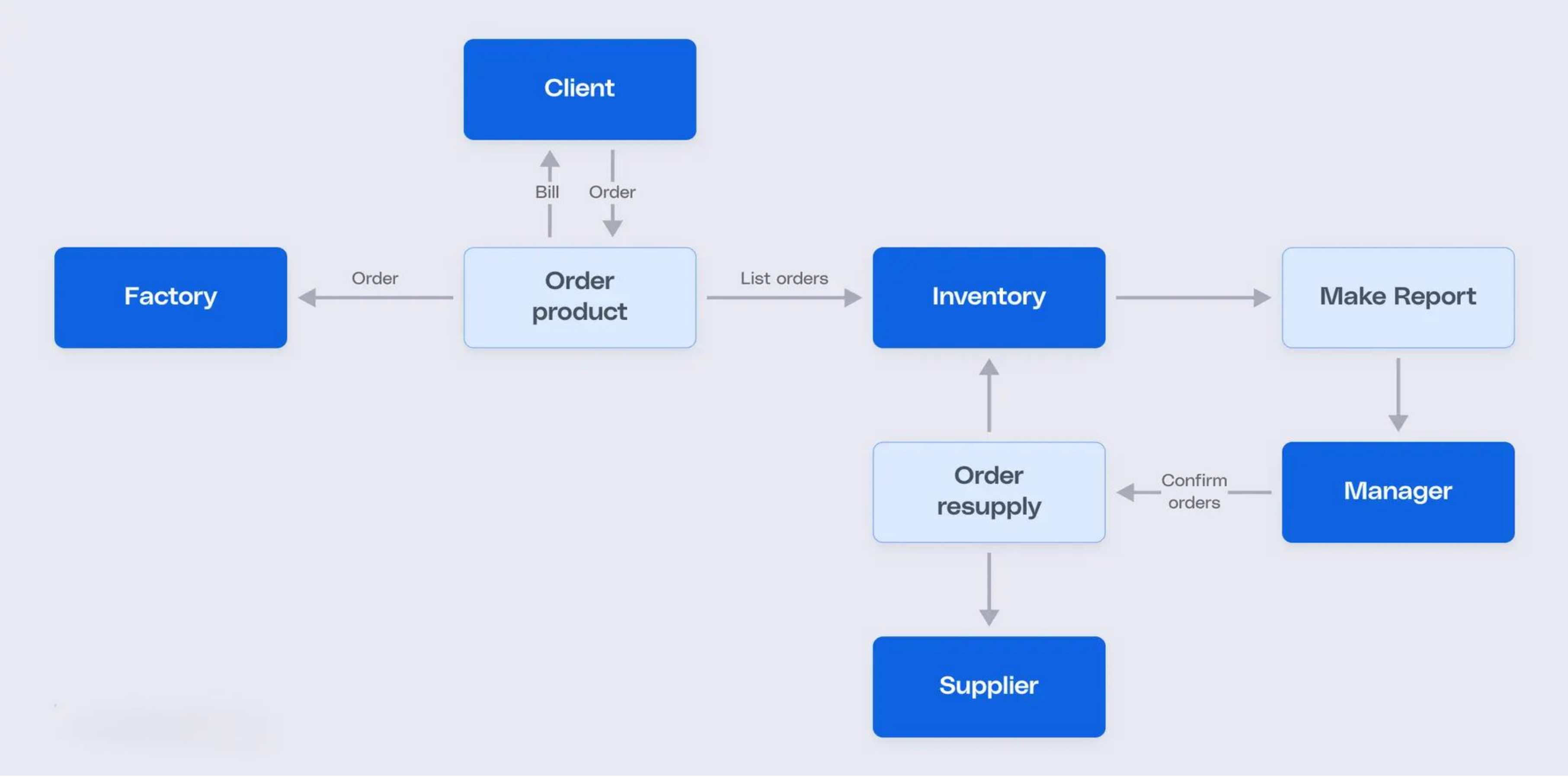
- A **use case diagram** might show actors like "Customer" and "Delivery Agent" interacting with features like "Place Order" or "Track Delivery".

- **Use Cases:** Login, Browse Products, Add to Cart, Checkout, Manage Orders
  - Customer → [Login]
  - Customer → [Browse Products]
  - Customer → [Add to Cart] → [Checkout]
  - Admin → [Manage Orders]

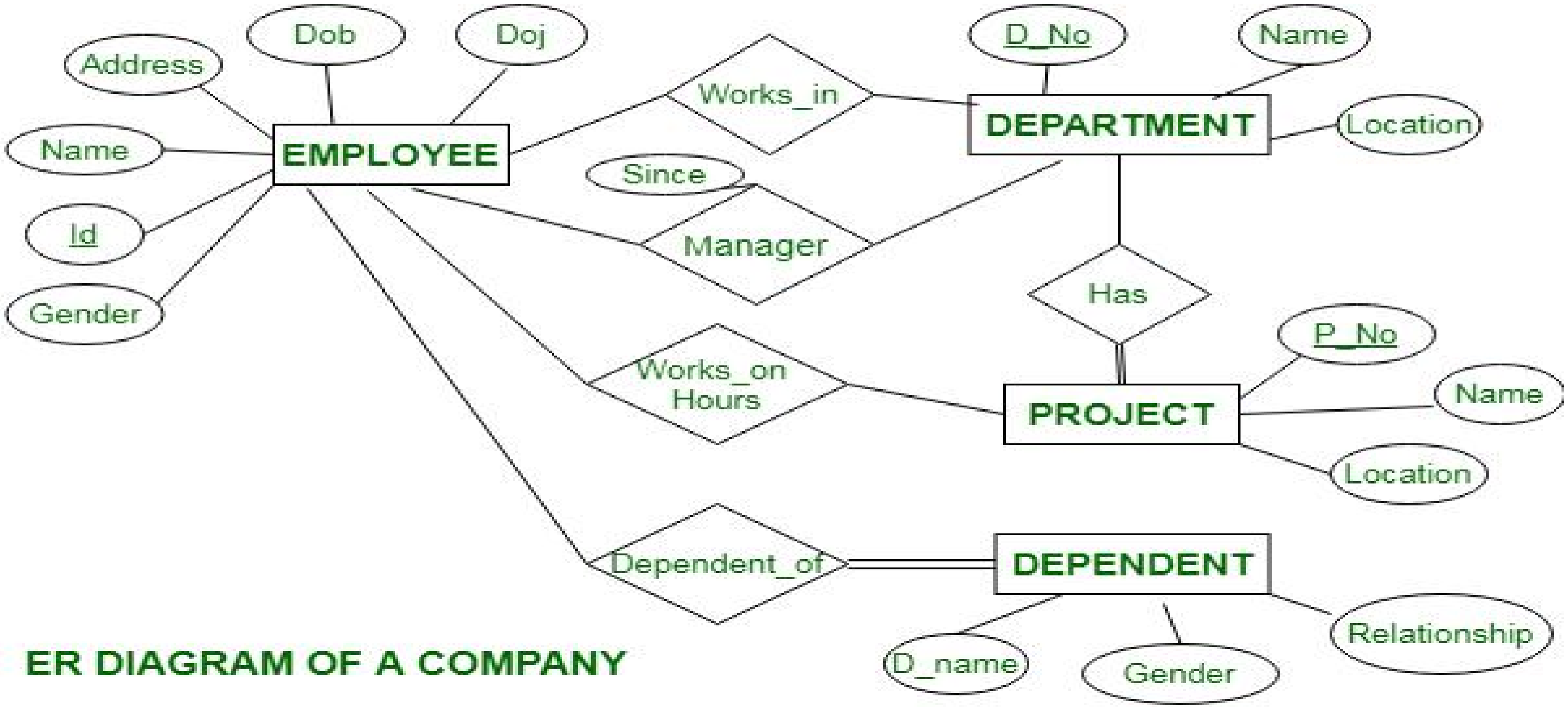
e-commerce website Use Case Diagram


Login
Add/Update/Delete Category
Add to Cart
Add/Update/Delete Product
Update/Delete Cart
Add/Update/Delete Manufacturer
Order Placement
Add Warehouse
Calculate Shipping and Tax
Add Country and State
Payment
Add/update shipping charge and tax
Logout
Actor
User
Paypal

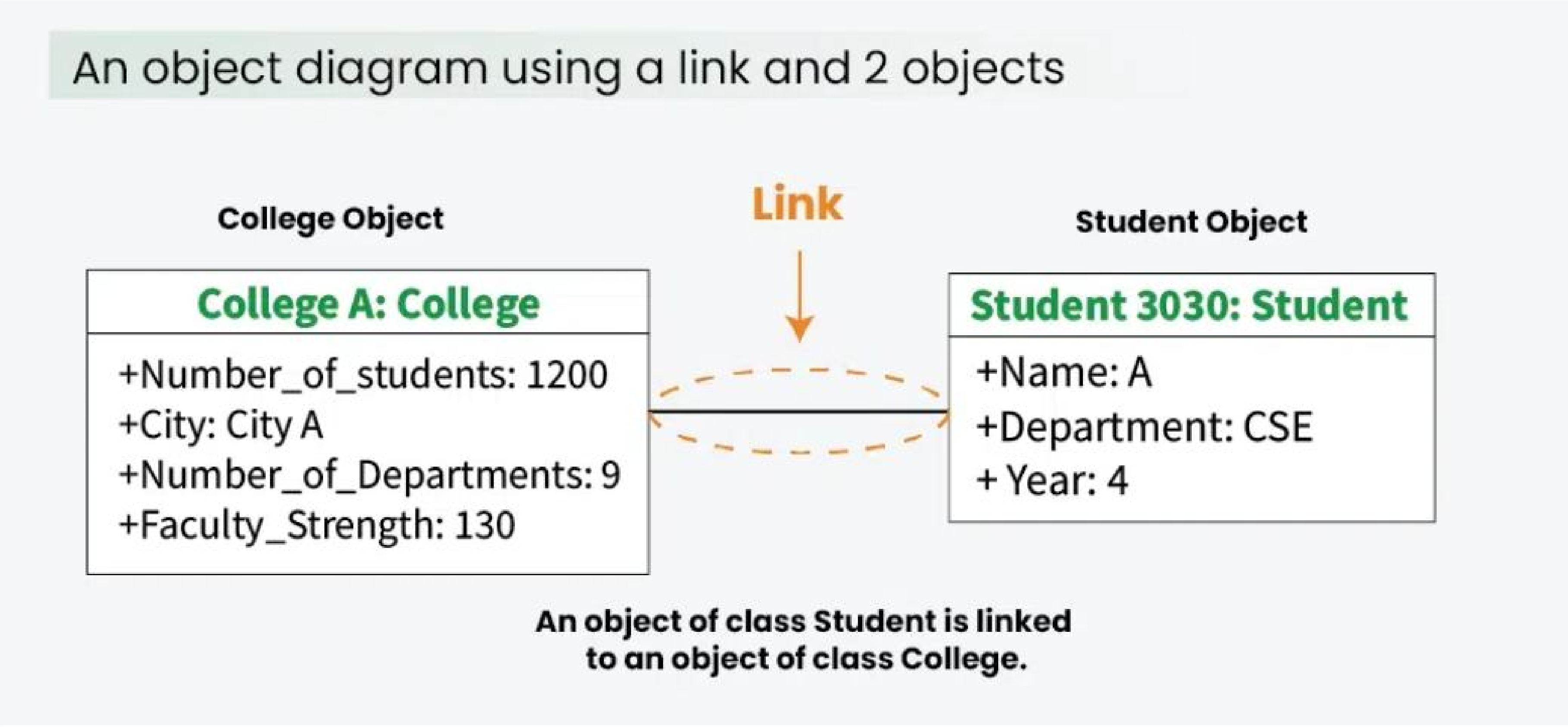
Data Flow Diagrams - Visualize how data moves through the system



Entity-Relationship Diagrams - Model data entities and their relationships



Class Diagrams (UML) - Represent system structure using classes and relationships



An object diagram using a link and 2 objects

College A: College

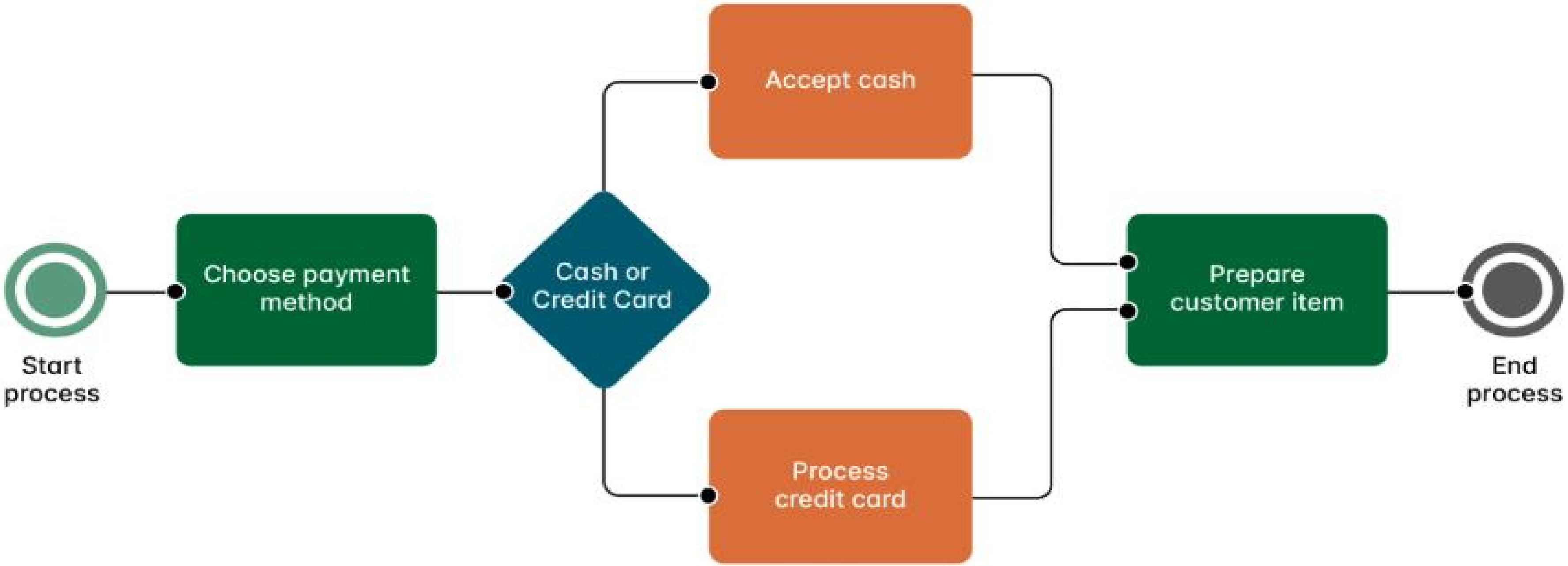
+Number_of_students: 1200
+City: City A
+Number_of_Departments: 9
+Faculty_Strength: 130

Student 3030: Student

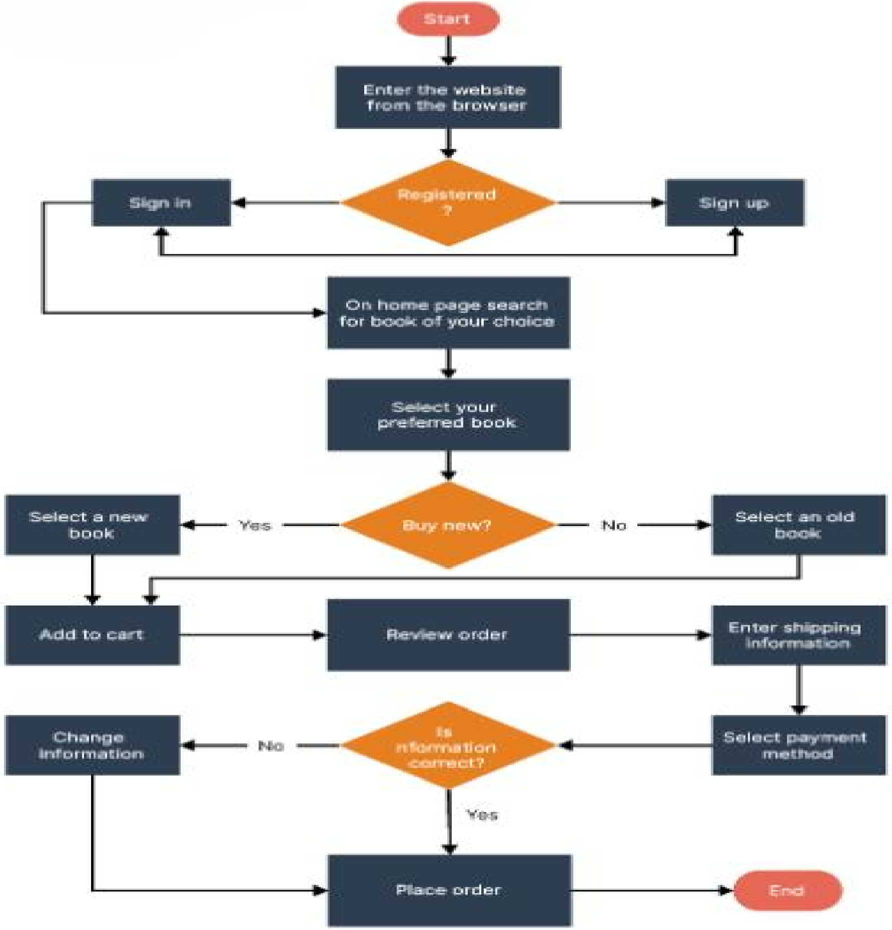
+Name: A
+Department: CSE
+ Year: 4

An object of class Student is linked
to an object of class College.

Business process modeling notation (BPMN)



Flowcharts - Show step-by-step logic or control flow



Start

Enter the website from the browser

Registered ?

Sign in

Sign up

On home page search for book of your choice

Select your preferred book

Buy new?

Yes

No

Select a new book

Select an old book

Add to cart

Review order

Enter shipping information

Select payment method

Is information correct?

No

Yes

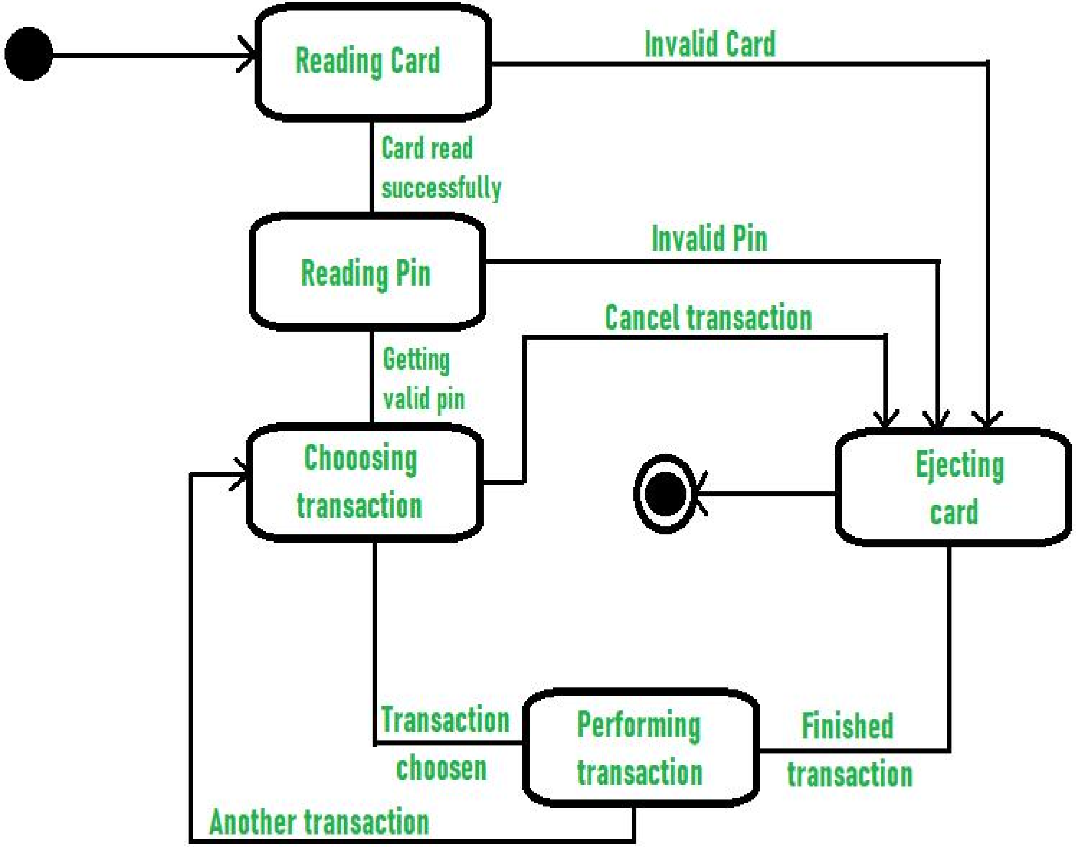
Change information

Place order

End

State Diagrams -

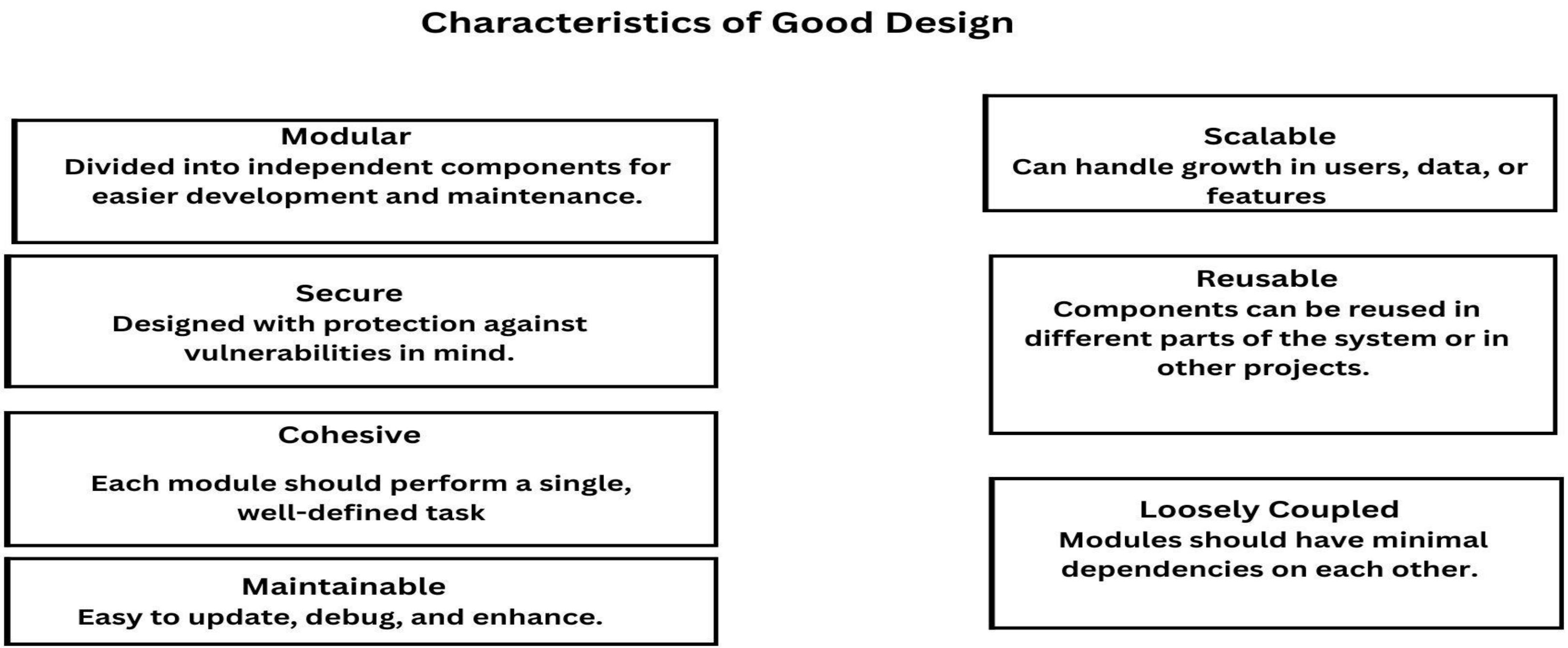
Describe how a system behaves in response to events



State Transition Diagram for ATM System

Design and Architectural Engineering

- This phase involves defining the **structure**, **components**, **interfaces**, and **data** for a software system to meet specified requirements. It includes both **high-level architecture** and **detailed design**.



Characteristics of Good Design

Modular
Divided into independent components for easier development and maintenance.

Secure
Designed with protection against vulnerabilities in mind.

Cohesive

Each module should perform a single, well-defined task

Maintainable
Easy to update, debug, and enhance.

Scalable
Can handle growth in users, data, or features

Reusable
Components can be reused in different parts of the system or in other projects.

Loosely Coupled
Modules should have minimal dependencies on each other.

Function Oriented System

- This focuses on identifying and modeling the functions or operations that a system performs.
- In this, the system is decomposed into a set of functions or modules, each of which performs a specific task.
- The functions are then interconnected to create the overall system.
- The main goal of function-oriented modeling is to create a system that is efficient, reliable, and easy to maintain.

Object Oriented System

- This focuses on identifying and modeling the objects that make up a system.
- In this, a system is decomposed into a set of objects, each of which has its own set of attributes and behaviors.
- The objects are then interconnected to create the overall system.
- The main goal of object-oriented modeling is to create a system that is modular, extensible, and reusable.

Function Oriented vs Object Oriented System

Aspect	Function-Oriented System	Object-Oriented System
Definition	Based on functions or procedures which operate on data.	Based on real-world entities using objects that combine data and behavior.
Focus	Focuses on functions (what the system does).	Focuses on objects (who is doing it).
Modularity	Divides the system into modules and functions.	Divides the system into objects and classes.
Data Handling	Data is global and can be accessed by any function.	Data is encapsulated within objects; access is controlled via methods.
Code Reusability	Limited reusability; functions are reused manually.	High reusability through inheritance and polymorphism.

Aspect	Function-Oriented System	Object-Oriented System
Security	Less secure due to global data sharing.	More secure due to encapsulation and access specifiers
Examples	C, Pascal, Fortran	Java, C++, Python
Design Approach	Top-down approach: problem is broken into smaller functions.	Bottom-up approach: system is built from objects and classes.
Data and Behavior	Data and behavior are separate	Data and behavior are bundled into objects.
Best Suited For	Small systems with limited complexity.	Complex and large-scale systems with evolving requirements.

Key Principles of Software Engineering

- Modularity – Divide into smaller parts
- Abstraction – Hide internal details
- Reusability – Use code in other projects
- Scalability – Handle growth in users/data
- Maintainability – Easy to fix or improve
- Documentation – Clear and up-to-date information

- **Modularity**: Dividing a system into distinct components (modules) that can be developed, tested, and maintained independently.
- **Cohesion**: Measures how closely related the functions within a single module are. High cohesion is desirable—it means the module does one thing well.
- **Coupling:** Describes how dependent modules are on each other. **Low coupling** is ideal—it makes systems more flexible and easier to maintain.
- **Layering**: Organizing software into layers (e.g., presentation, business logic, data access) to separate concerns and improve scalability.

Ideal Design: *High cohesion + Low coupling → Strong modularity*

Modularity

- Dividing a system into distinct components (modules) that can be developed, tested, and maintained independently.

Real-World Applications

- **Microservices architecture**: Each service is a module with a specific responsibility
- **Plugin systems**: Software like IDEs use modular plugins to extend functionality
- **Library-based development**: Developers use modular libraries for tasks like authentication, logging, etc.

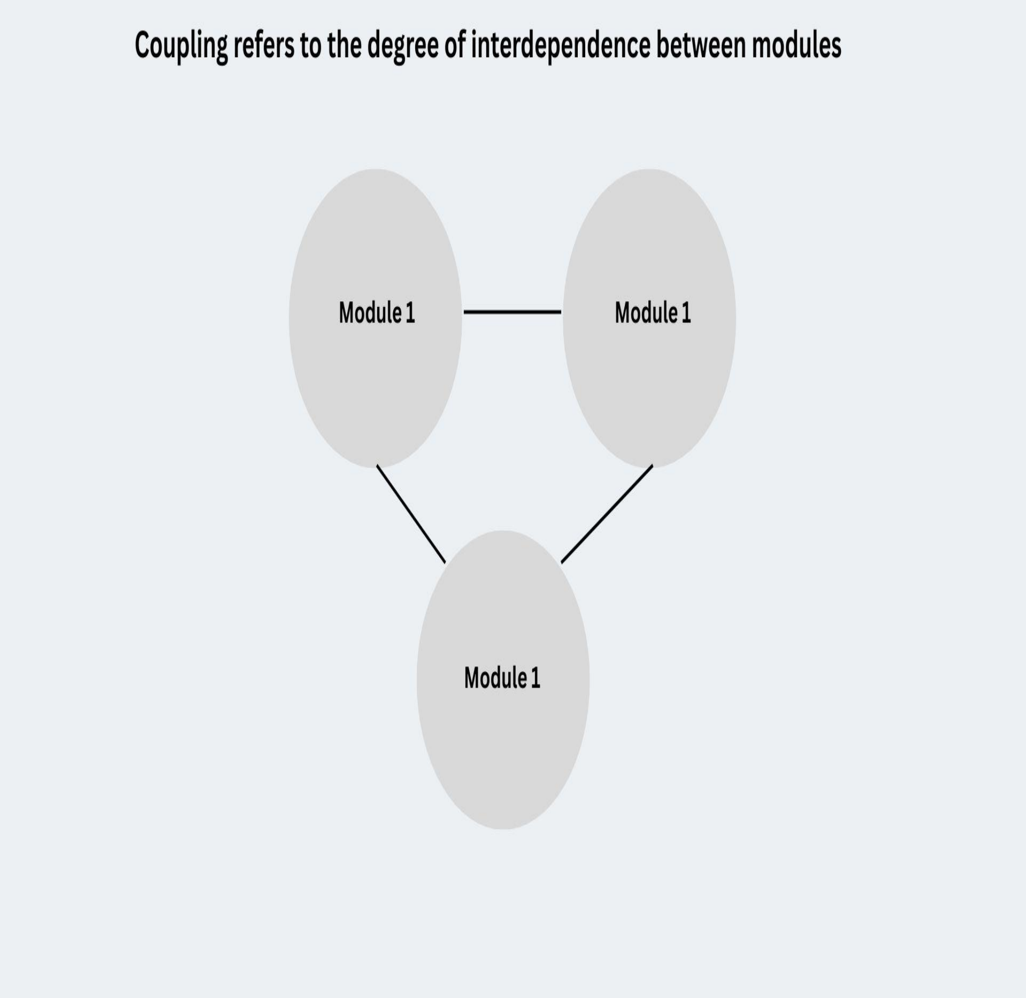
Cohesion

- Measures how closely related the functions within a single module are.
- High cohesion is desirable—it means the module does one thing well.
- Cohesion is categorized into several types, ranked from worst to best:

Type of Cohesion	Description
Coincidental	Elements are grouped arbitrarily; no meaningful relationship
Logical	Elements perform similar activities but are selected via control statements
Temporal	Elements are related by timing (e.g., initialization tasks)
Procedural	Elements follow a specific sequence of execution
Communicational	Elements operate on the same data
Sequential	Output from one part is input to another
Functional	All elements contribute to a single, well-defined task

Coupling

- Describes how dependent modules are on each other.
- High coupling means that modules are closely connected and changes in one module may affect other modules
- Low coupling means that modules are independent, and changes in one module have little impact on other modules.
- So it is ideal and makes systems more flexible and easier to maintain.

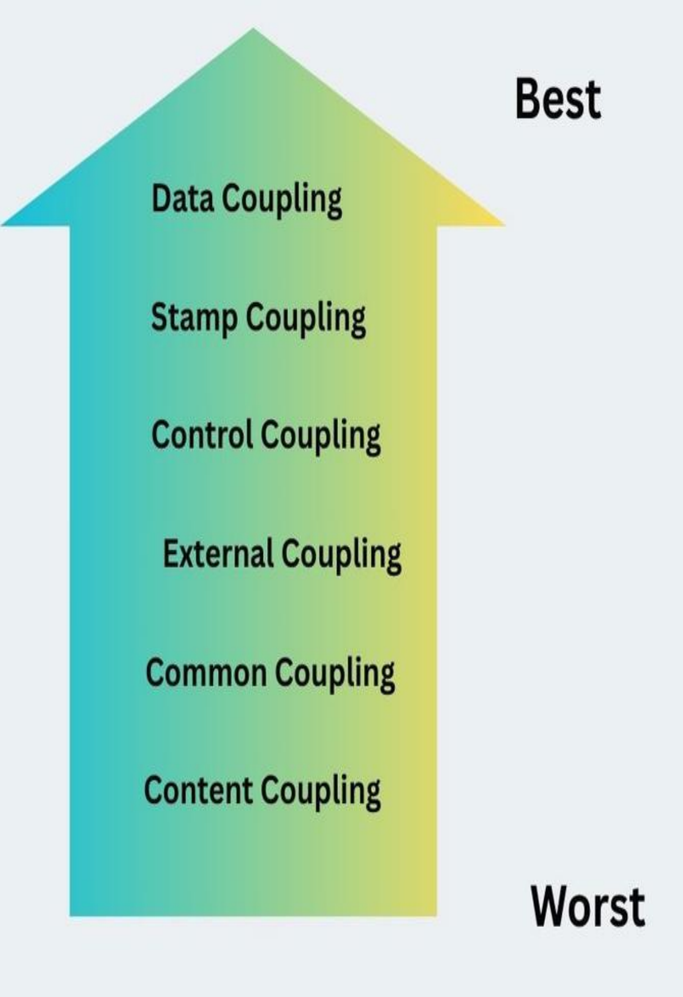


Coupling refers to the degree of interdependence between modules

Types of Coupling

Data Coupling: If the dependency between the **modules** is based on the fact that they **communicate by passing only data**, then the modules are said to be data coupled.

- In data coupling, the components are independent of each other and communicate through data.
- Module communications don't contain tramp data.
- **Example**-customer billing system.



Stamp Coupling

- In stamp coupling, the complete data structure is passed from one module to another module.
- Therefore, it involves tramp data.
- It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.

Control Coupling

- If the modules communicate by passing control information, then they are said to be control coupled.
- It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality.
- **Ex**- sort function that takes comparison function as an argument.

External Coupling

- In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware.
- **Ex**- protocol, external file, device format, etc.

Common Coupling

- The modules have shared data such as global data structures.
- The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change.
- So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.

Content Coupling

- In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module.
- This is the worst form of coupling and should be avoided.

Design Models

- Design models describe how the system will fulfill requirements.
- They bridge the gap between analysis and coding.

  Common models include:
  - Structural Models: Class diagrams, object diagrams.
  - Behavioral Models: Sequence diagrams, activity diagrams.
  - Architectural Models: High-level structure of the system (e.g., MVC, 3-tier).
  - Data Models: Describe how data is stored and accessed.
- These models help visualize, verify, and refine the system architecture before coding.
- **Data Flow Diagrams (DFDs)**: Show how data moves through the system. Entity-Relationship Diagrams (ERDs): Model data and relationships.
- **State Diagram**s: Represent system behavior based on events and states.

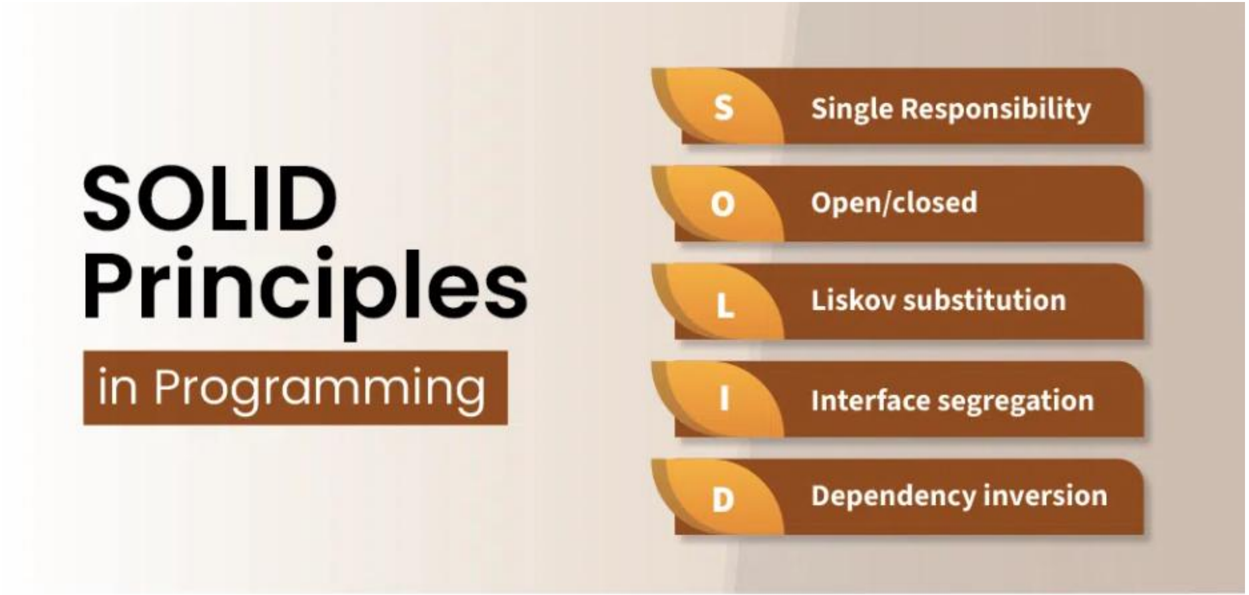
UML (Unified Modeling Language)

- UML is a standardized modeling language for visualizing and documenting software design in object-oriented development.
- Common UML diagrams include:
  - Structural Diagrams: Class, Object, Component, Deployment.
  - Behavioral Diagrams: Use Case, Activity, Sequence, State.
  - Interaction Diagrams: Sequence and Collaboration diagrams.
  - Use Case Diagrams: Capture system functionality from a user’s perspective.
  - Sequence Diagrams: Detail object interactions over time.
- UML is widely used in Object-Oriented Analysis and Design to communicate system structure and behavior clearly.

Coding

Programming Principles

- These include SOLID principles, DRY (Don’t Repeat Yourself), and KISS (Keep It Simple, Stupid)—Style guides and naming rules that ensure consistency across a codebase (e.g., camelCase for variables, indentation rules, etc.).
- These principles enhance code readability, reusability, and maintainability.
- All aimed at writing clean, maintainable code.



SOLID Principles in Programming

SOLID Principles

- **Single Responsibility Principle**

  Ex - Imagine a baker who is responsible for baking bread. The baker's role is to focus on the task of baking bread, ensuring that the bread is of high quality, properly baked, and meets the bakery's standards.

- **Open/Closed Principle**

  Ex - Imagine you have a class called PaymentProcessor that processes payments for an online store. Initially, the Payment Processor class only supports processing payments using credit cards. However, you want to extend its functionality to also support processing payments using PayPal.

- **Liskov's Substitution Principle**

  Ex - One of the classic examples of this principle is a rectangle having four sides. A rectangle's height can be any value and width can be any value. A square is a rectangle with equal width and height. So we can say that we can extend the properties of the rectangle class into square class.

- **Interface Segregation Principle**

  Ex - Suppose if you enter a restaurant and you are pure vegetarian. The waiter in that restaurant gave you the menu card which includes vegetarian items, non-vegetarian items, drinks, and sweets.

SOLID Principles

Dependency Inversion Principle

- In a software development team, developers depend on an abstract version control system (e.g., Git) to manage and track changes to the codebase. They don't depend on specific details of how Git works internally.

Coding Conventions

- These are guidelines for writing consistent and readable code, including:
  - Naming conventions (e.g., camelCase, PascalCase)
  - Proper indentation and spacing
  - Commenting and documentation
  - Avoiding hard-coded values
  - Error handling

- Coding conventions **ensures teamwork efficiency and reduces bugs.**

Object Oriented Analysis and Design

- It is a methodology that analyzes and designs software using object-oriented concepts
- It identifies system requirements and models them as objects (real-world entities).
- It defines how these objects interact and are implemented in code.
- This approach focuses on modeling a system as a group of interacting objects:
    - Analysis: Identifies the objects and their relationships based on requirements.
    - Design: Defines how these objects will interact and be implemented using principles like encapsulation, inheritance, and polymorphism.

Key OOAD Concepts:

- Classes and Objects
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

- OOAD is effective for building scalable, reusable, and maintainable systems.

Sessions 6, 7 & 8

- Introduction to Agile development model
- Agile development components
- Benefits of Agile
- Introduction to different tools used for agile web development
- Scrum and Extreme Programming
- Introduction to Atlassian Jira
  - Add Project
  - Add Tasks and sub-tasks
  - Create sprints with tasks
- Case study of developing web application using agile methodology

Introduction to Agile Development Model

- The Agile development model is a **flexible approach** to software development that emphasizes collaboration, customer feedback, and rapid delivery of small, functional pieces of a project.
- Unlike traditional models like the Waterfall, Agile is **iterative and incremental**, meaning software is built in small parts (called iterations or sprints) and continuously improved based on user feedback.

Key Features of Agile Model:

Iterative Approach

- Work is divided into small cycles (sprints), usually 1–4 weeks long.
- After each cycle, a working part of the product is delivered.

Customer Involvement

- Clients are involved throughout the project.
- Their feedback is taken after each iteration, ensuring the product meets their needs.

Cross-Functional Teams

- Agile teams are small, self-organizing, and include developers, testers, designers, and product owners working together.

Introduction to Agile Development Model

Adaptive Planning

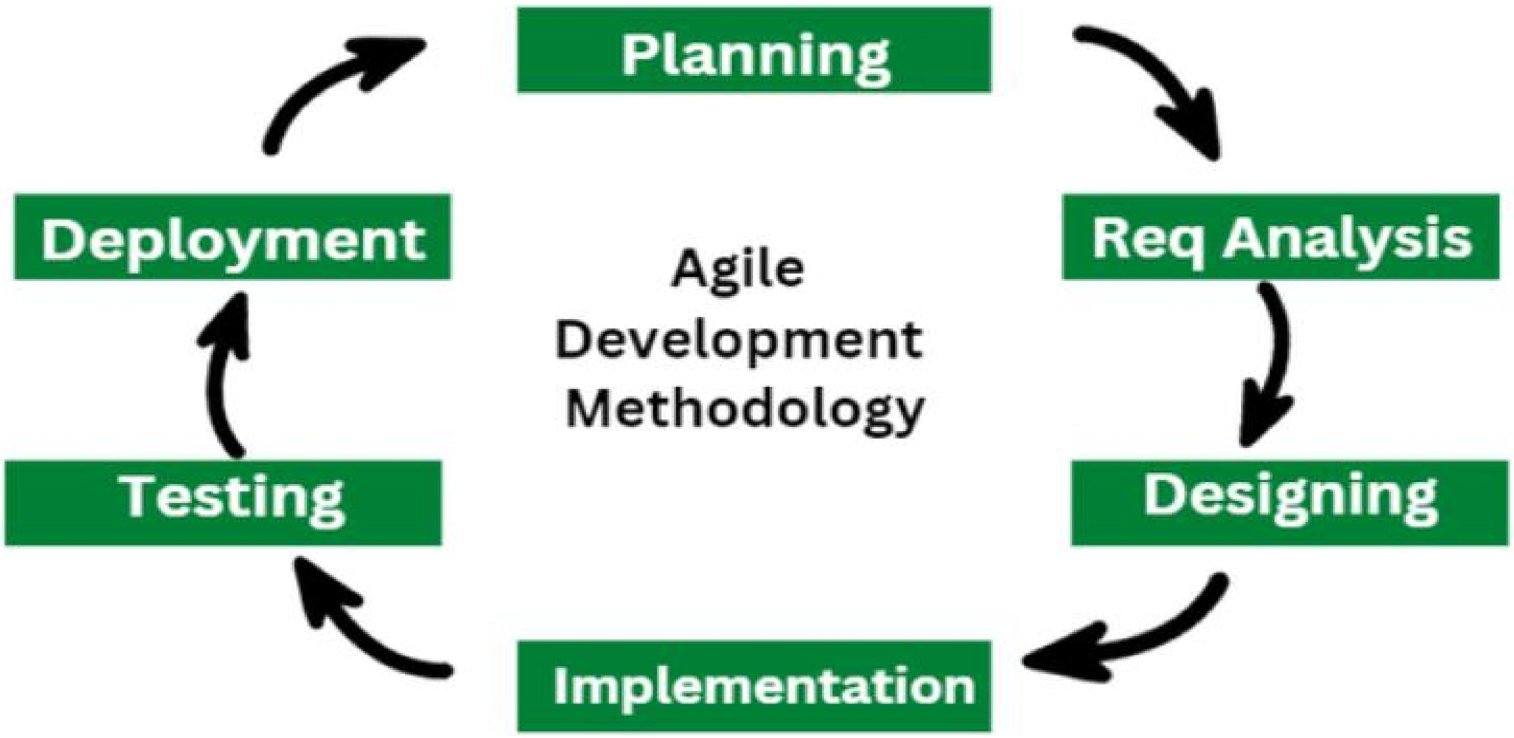
- Agile supports changing requirements even in the late stages of development.
- This makes it flexible and responsive.

Continuous Improvement

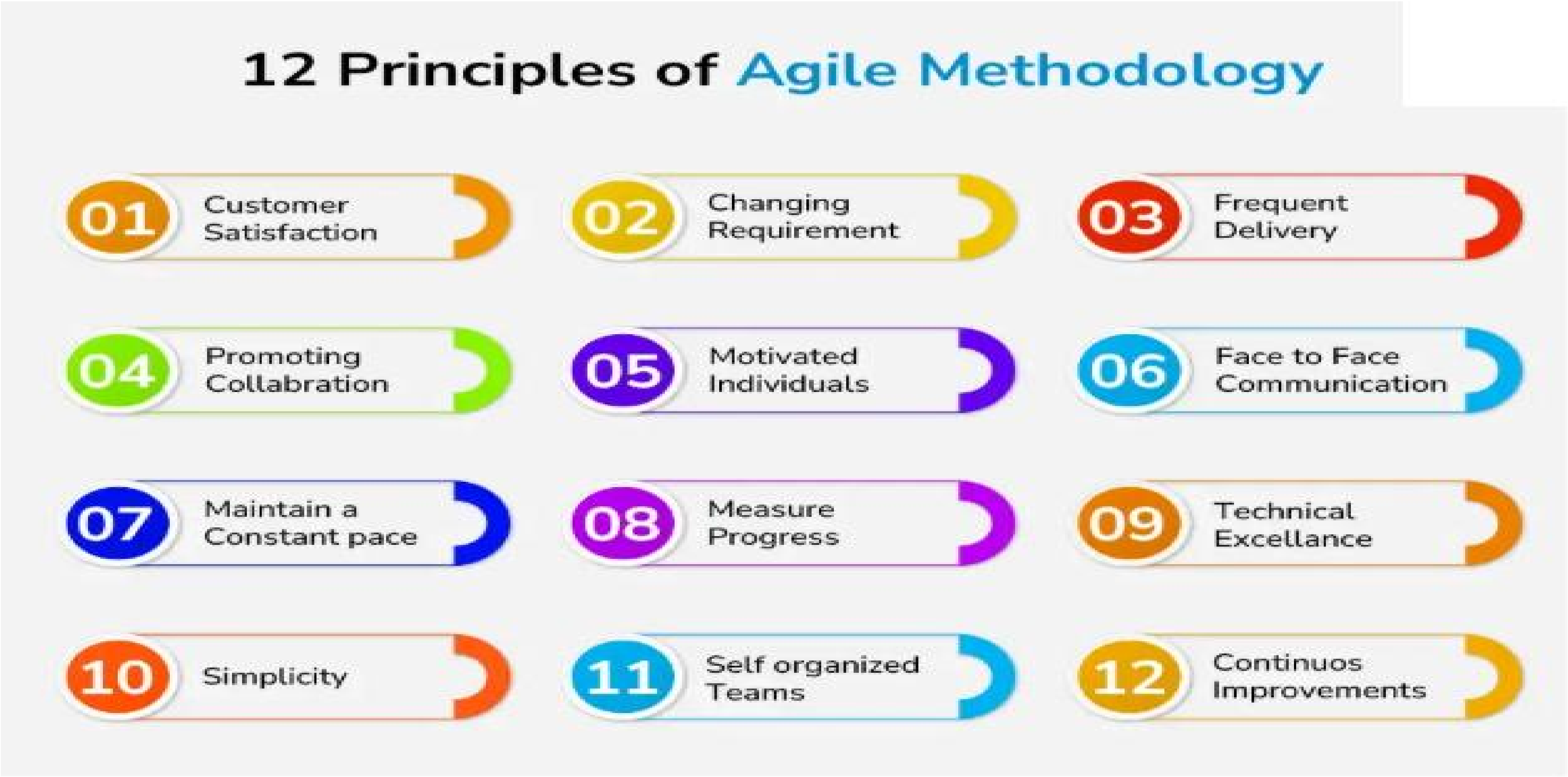
- After each sprint, teams hold a retrospective to identify what went well and what needs improvement.

Working Software as the Main Measure

- Agile focuses more on delivering working software frequently rather than waiting for full c0



Agile Development Stages



12 Principles of Agile Methodology

01 Customer Satisfaction

02 Changing Requirement

03 Frequent Delivery

04 Promoting Collabration

05 Motivated Individuals

06 Face to Face Communication

07 Maintain a Constant pace

08 Measure Progress

09 Technical Excellance

10 Simplicity

11 Self organized Teams

12 Continuos Improvements

12 agile principles

Agile Development Components

These below components define how Agile teams plan, execute, and deliver software effectively.

1. User Stories

- These are short, simple descriptions of a feature told from the perspective of the user.
- **Ex**: *"As a user, I want to reset my password so I can regain access to my account."*
- Helps focus development on real user needs.

2. Product Backlog

- A list of all features, changes, bug fixes, and tasks needed for the product.
- Maintained by the Product Owner.
- Items in the backlog are prioritized based on business value.

3. Sprint (or Iteration)

- A time-boxed development cycle, typically 1 to 4 weeks.
- A fixed set of tasks is chosen from the backlog and implemented.
- At the end of the sprint, working software is delivered.

4. Daily Stand-up (Daily Scrum)

- A short daily team meeting (usually 15 minutes).
- Team members discuss:
  - What they did yesterday
  - What they will do today

5. Sprint Planning

- Meeting held at the beginning of each sprint.
- The team decides which user stories from the backlog will be completed in the sprint.

6. Sprint Review

- Conducted at the end of a sprint.
- The team demonstrates the completed features to stakeholders for feedback.

7. Sprint Retrospective

- Team discusses what went well, what didn’t, and how to improve in future sprints.
- Encourages continuous improvement.

8. Agile Roles

- Product Owner: Defines requirements and prioritizes the backlog.
- Scrum Master: Facilitates the Agile process and removes obstacles.
- Development Team: Builds the product increment.

9. Increment

- The usable and shippable part of the product delivered at the end of a sprint.
- Each increment adds value and moves the project forward.

10. Definition of Done (DoD)

- A checklist of criteria that must be met before a product backlog item is considered "done".

Agile Development Components

- User Stories
- Product Backlog
- Sprint (or Iteration)
- Daily Scrum
- Sprint Planning
- Sprint Review
- Sprint Retrospective
- Agile Roles
- Increment
- Definition of Done (DoD)

Benefits of Agile

The Agile development model offers numerous advantages that make it one of the most popular software development methodologies today. It focuses on flexibility, speed, collaboration, and customer satisfaction.

1. Flexibility & Adaptability - Agile thrives in dynamic environments. It allows teams to respond quickly to changing requirements, even late in the development cycle.
2. Faster Time-to-Market - By delivering work in small, functional increments, Agile ensures quicker releases and faster feedback loops.
3. Improved Product Quality - Continuous testing and integration throughout the development process lead to fewer bugs and a more polished final product.
4. Higher Customer Satisfaction - Frequent collaboration and regular demos keep customers engaged and ensure the product aligns with their expectations.
5. Enhanced Team Collaboration - Agile promotes open communication, daily stand-ups, and shared ownership, which boosts morale and productivity.
6. Better Risk Management - Regular reviews and retrospectives help identify issues early, reducing the chance of major failures.
7. Continuous Improvement - Agile encourages teams to reflect and refine their processes after each sprint, fostering a culture of learning and growth.

Goal of Agile Development

- Agile development brings flexibility, efficiency, and customer-focused results.
- Its iterative nature helps teams deliver **high-quality products faster**, while constantly learning and improving.

Introduction to Tools Used in Agile Web Development

- Agile web development is a fast-paced and collaborative process that involves frequent code updates, continuous testing, and quick feedback.
- To support this, developers use a wide range of Agile tools that help in planning, tracking, collaboration, integration, testing, and deployment.

1. Project Management & Tracking Tools
    - Jira: Widely used Agile tool for creating user stories, sprint planning, and tracking progress with Scrum or Kanban boards.
    - Trello: A lightweight task board using cards and lists for managing workflows.
    - Asana: Task and project tracking with timeline views, suited for collaborative teams.
    - ClickUp / Monday.com: All-in-one project management tools supporting Agile boards, sprints, and team goals.

2. Version Control Tools : They help manage changes in code and enable team collaboration.

- Git: Most popular version control system; developers can work on code independently.
- GitHub / GitLab / Bitbucket: Platforms built on Git, offering source code hosting, pull requests, issue tracking, and collaboration features

3. Continuous Integration / Continuous Deployment (CI/CD) Tools : These automate testing, building, and deploying code.

- Jenkins: Open-source automation server for building, testing, and deploying code.
- CircleCI / Travis CI: Cloud-based CI/CD tools that integrate with GitHub.
- GitLab CI/CD: Built-in pipeline for GitLab repositories.

4. Communication & Collaboration Tools : Essential for daily stand-ups, sprint meetings, and team discussions.

- Slack: Real-time messaging platform with integrations for Jira, GitHub, etc.
- Microsoft Teams: Offers chat, meetings, and collaboration in one place.
- Zoom / Google Meet: Used for remote sprint meetings and reviews.

5. Testing Tools : Automated testing ensures bug-free code in each iteration.

- Selenium: Popular for browser automation and web app testing.
- Postman: API testing tool used for checking backend services.
- JUnit / Mocha / Jasmine: Used for unit testing in JavaScript, Java, etc.

Scrum and Extreme Programming

- Scrum and Extreme Programming (XP) are both Agile frameworks, but they focus on different aspects of software development.
- Think of Scrum as the **project management** side of Agile, while XP is all about **engineering excellence**.

Scrum: Managing the Process

- Focus - Organizing work and teams
- Key Roles - Product Owner, Scrum Master, Development Team
- Structure - Time-boxed sprints (2–4 weeks), sprint planning, daily stand-ups, reviews, and retrospectives
- Flexibility - Once a sprint starts, changes are discouraged
- Engineering Practice - Not prescribed—teams choose their own

Extreme Programming (XP): Engineering Discipline

- Focus - Writing high-quality code through best practices
- Key Practices - Test-Driven Development (TDD), Pair Programming, Continuous Integration, Refactoring
- Structure - Short iterations (1–2 weeks), with flexibility to swap features mid-iteration if not yet started
- Flexibility - More adaptable to change within iterations
- Engineering Practices - Strongly emphasized and required

How Scrum and Extreme Programming Work Together?

- Many teams actually combine Scrum and XP—using Scrum to manage the workflow and XP to ensure technical quality.
- This hybrid approach leverages the strengths of both frameworks.

Introduction to Atlassian Jira

- Jira is a powerful Agile project management tool developed by Atlassian.
- It is widely used by software development teams to plan, track, and manage their work using Agile methodologies like Scrum and Kanban.
- It provides custom dashboards, detailed reports, and integrates with tools like GitHub, Bitbucket, Slack, Confluence, and more.

Jira allows teams to:

- Create projects
- Organize tasks and sub-tasks
- Track bugs and user stories
- Manage sprints and workflows

1. **Add Project in Jira**:
    - Login to Jira with your credentials.
    - On the dashboard, click on “Projects” → “Create Project”.
    - Choose a template (e.g., Scrum, Kanban, Bug Tracking).
    - Enter project details: name, key, and type.
    - Click “Create”.
    - The new project is now available for adding issues (tasks, stories, bugs).

2. **Add Tasks and Sub-Tasks** : In Jira, tasks are referred to as Issues, and they can be of various types like Story, Bug, Task or Epic.

   To Add a Task:

   - Go to your project.
   - Click “Create” on the top bar.
   - Select the issue type as “Task” or “Story”.
   - Fill in the details: Summary, Description, Assignee, Priority.
   - Click “Create”.

   To Add a Sub-Task:

   - Open the main task or story.
   - Click “More actions” → “Create Sub-task”.
   - Enter sub-task details and click Create.
   - Sub-tasks appear under the parent issues

3. **Create Sprints with Tasks (Scrum) :** In Scrum projects, Sprints help break the project into manageable time-boxed Iterations.

   To Create a Sprint:

   - Go to the “Backlog” view of your Scrum project.
   - Click on “Create Sprint”.
   - Drag and drop tasks (issues) from the backlog into the sprint.
   - Click “Start Sprint” and set: Sprint name Duration (1 to 4 weeks) - Start and end dates
   - Sprint is now active and visible on the Scrum board.
   - During the sprint, tasks are moved across stages like To Do → In Progress → Done.

case study of developing web application using agile methodology