Software Engineering & DevOps

- 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]

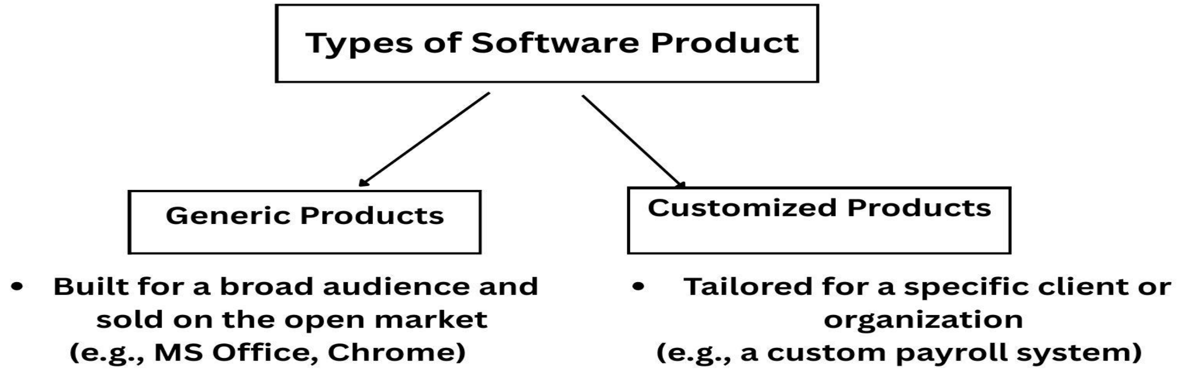
Common Software Process Model

Model	Description	Used For
Waterfall	Linear and sequential. Each phase must be completed before the next begins.	Small, well-defined projects
Agile	Iterative and flexible. Emphasizes collaboration and adaptability.	Dynamic projects with changing requirements
Spiral	Combines iterative development with risk analysis.	Large, high-risk projects
V-Model	Each development phase has a corresponding testing phase.	Projects requiring strong testing
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
Customized Products
• Built for a broad audience and sold on the open market (e.g., MS Office, Chrome)
• 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

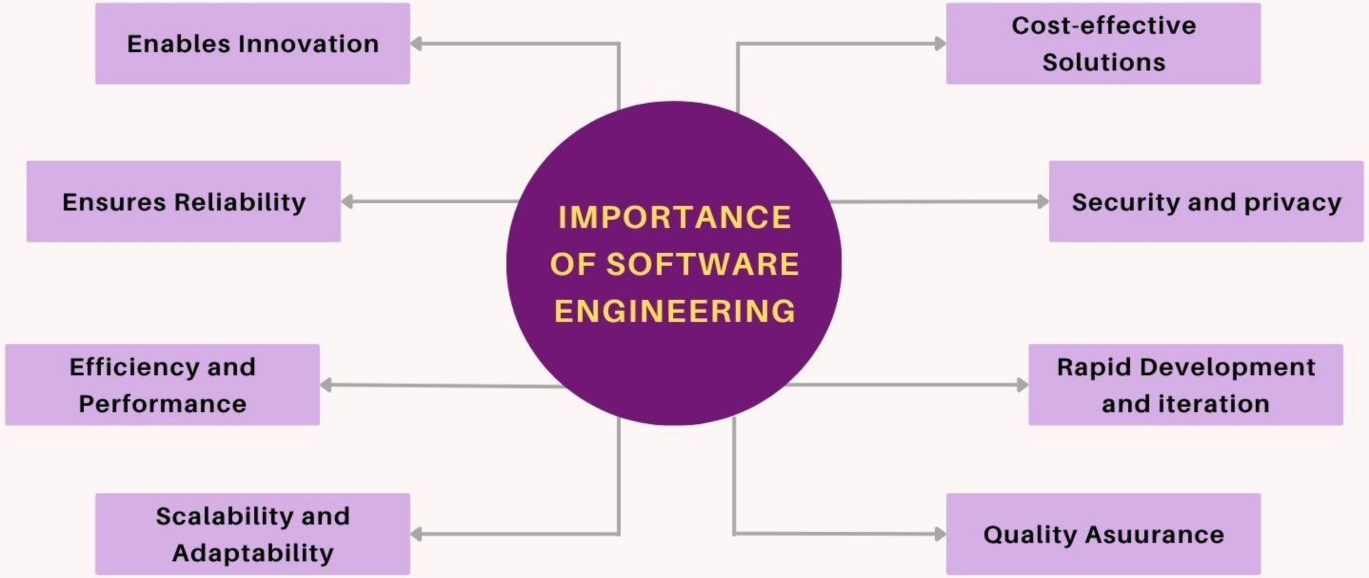
Test the initial product to identify issues and improvements.

Improvement & Refinement

Iterate based on feedback from product testing results.

Launch

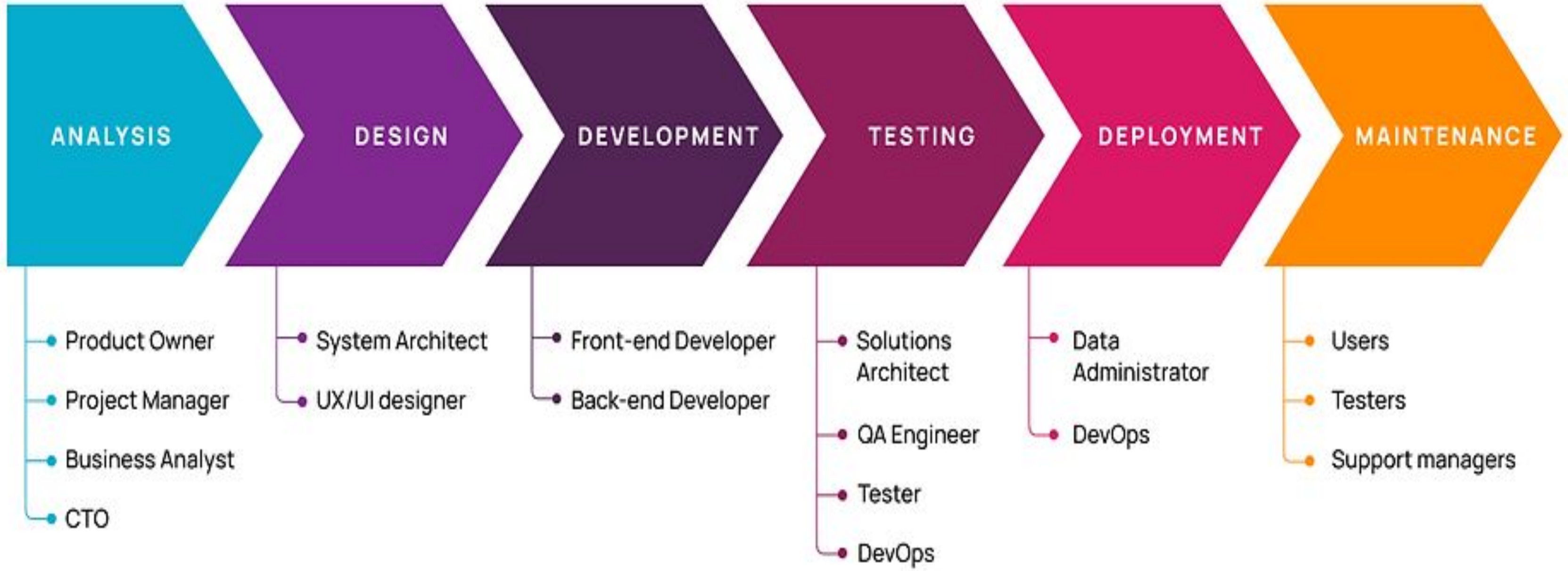
The product is launched to the market with an integrated marketing strategy.


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

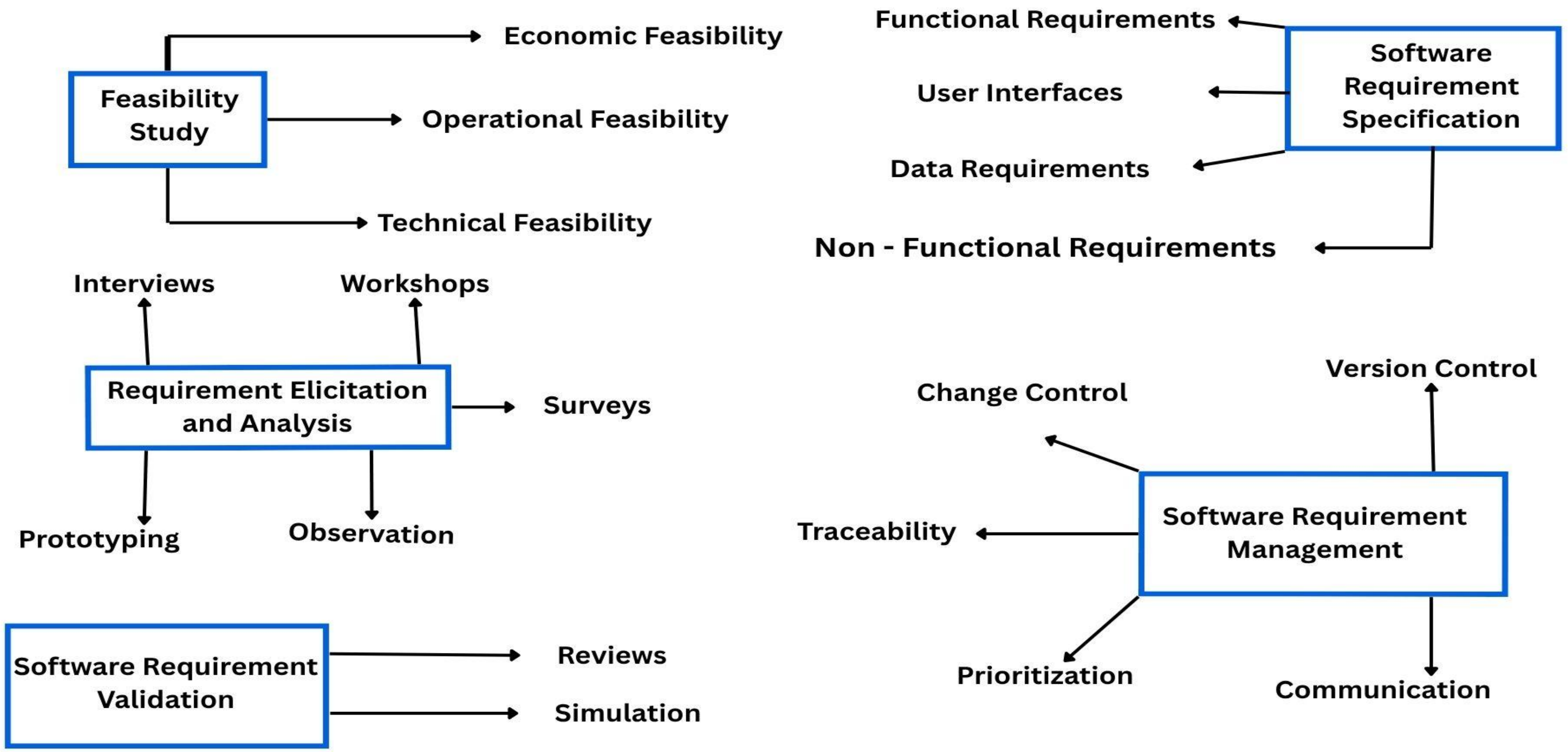
CHARACTERISTICS OF SOFTWARE ENGINEERING

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

6 Phases of the Software Development Life Cycle



Requirements Engineering



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
2) **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
3) **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

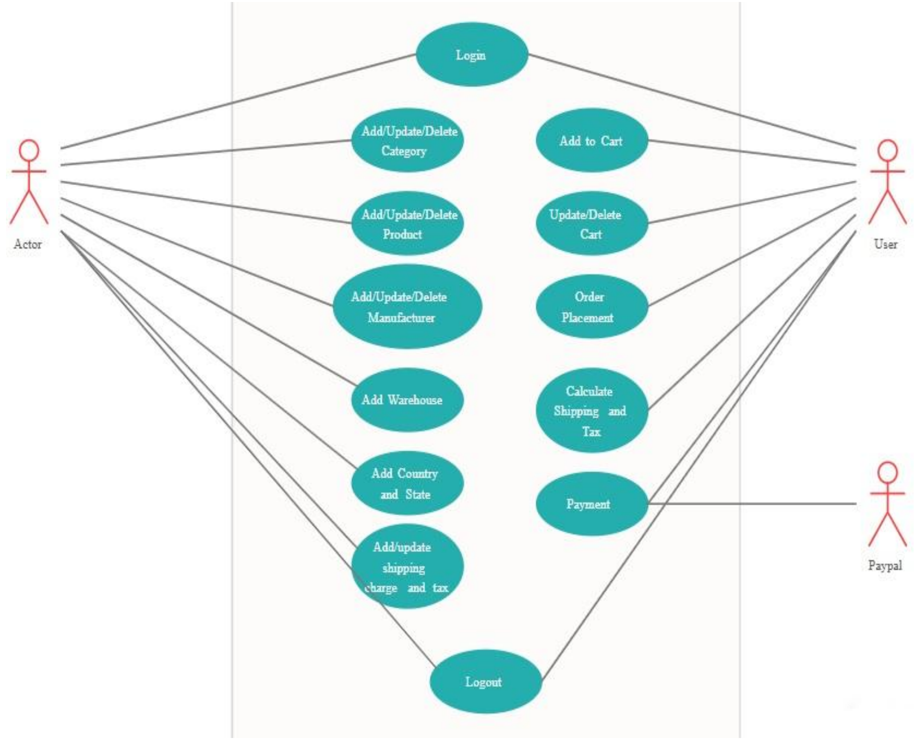
Technique	Description
Use Case Diagrams (UML)	Show interactions between users (actors) and the system
Data Flow Diagrams	Visualize how data moves through the system
Entity-Relationship Diagrams	Model data entities and their relationships
Class Diagrams (UML)	Represent system structure using classes and 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

e-commerce website Use Case Diagram

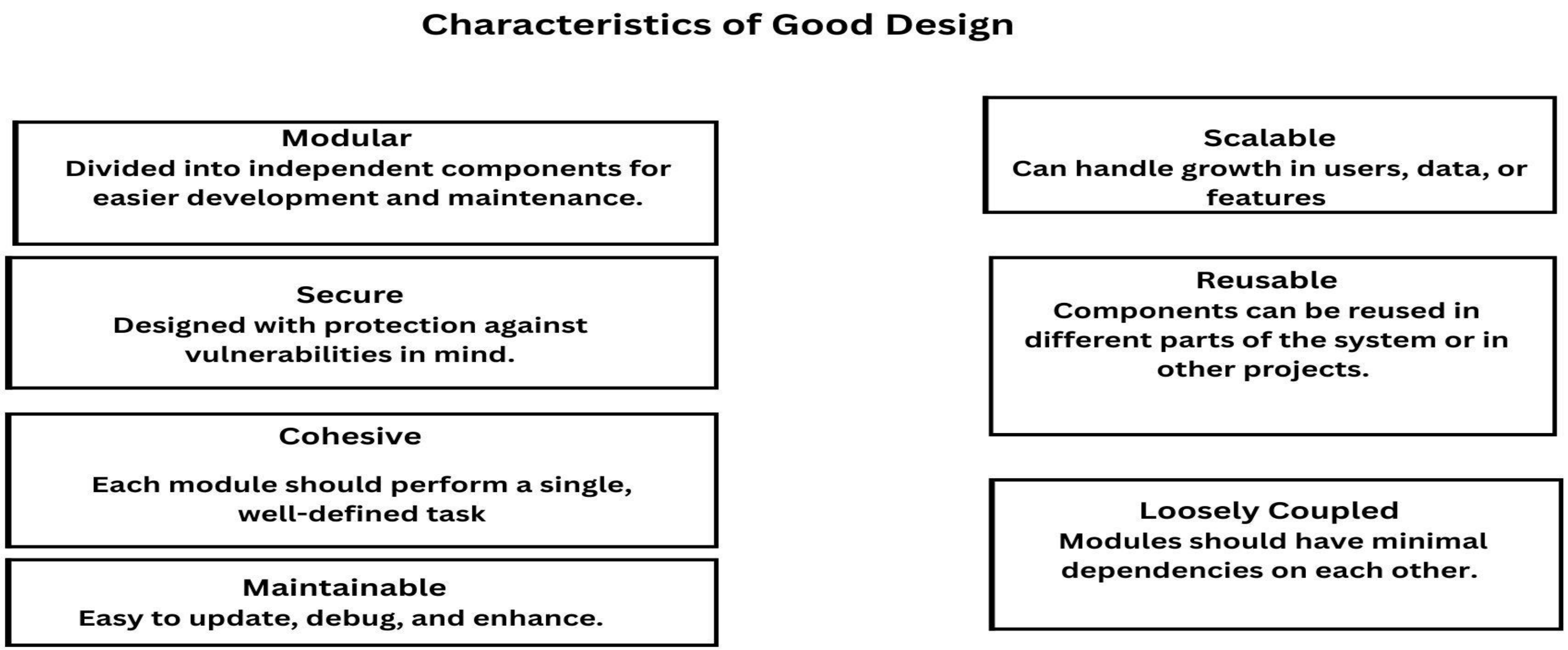
Imagine you're building an online food delivery app:

- 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]


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

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.

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.

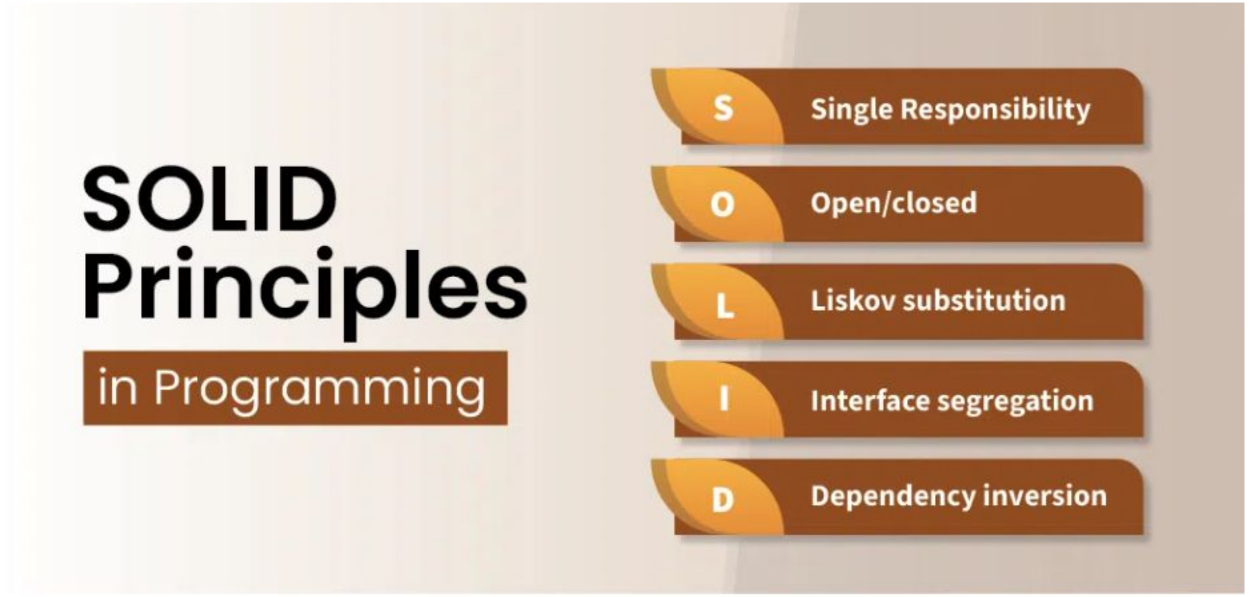
UML (Unified Modeling Language)

- 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

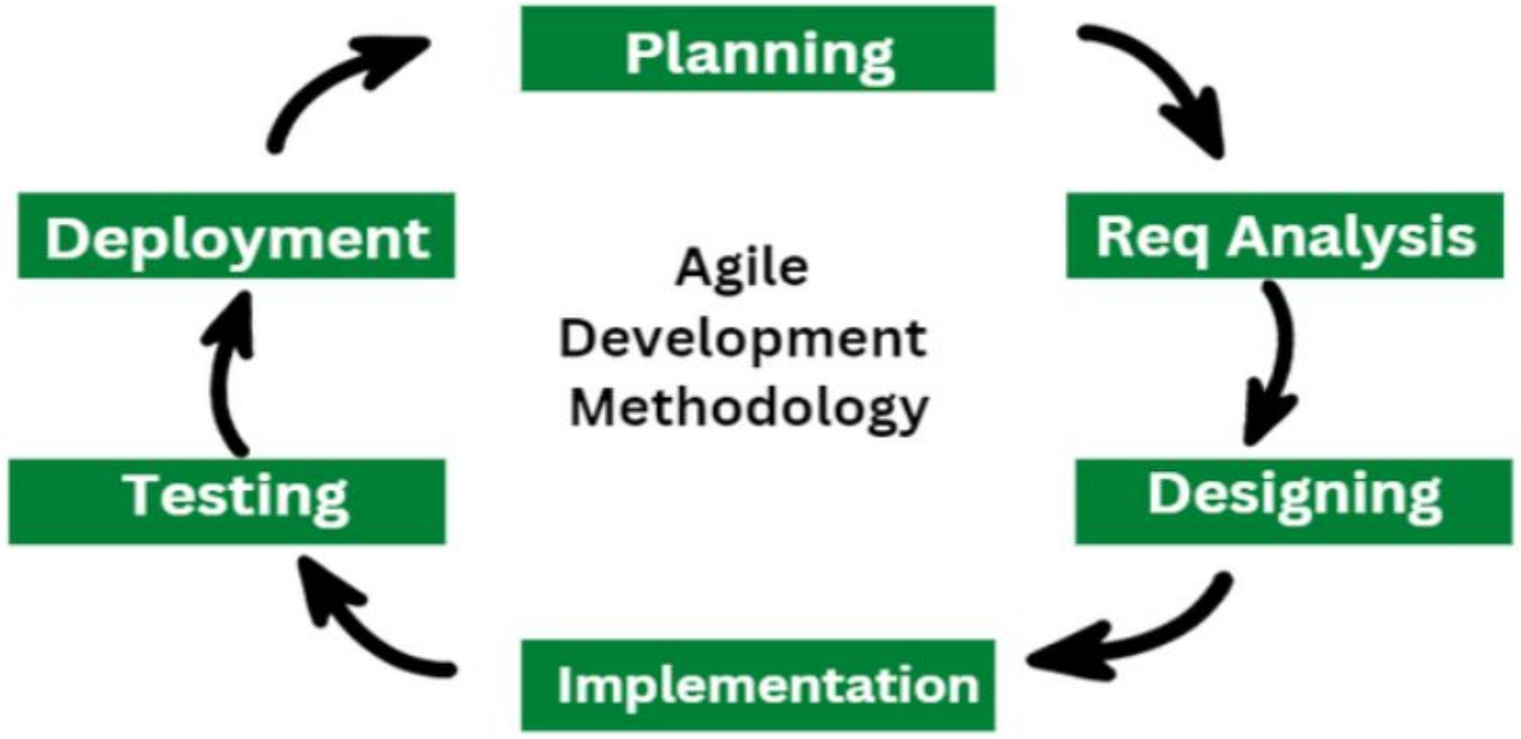
- Agile supports changing requirements even in the late stages of development.
- This makes it flexible and responsive.

Continuous Improvement

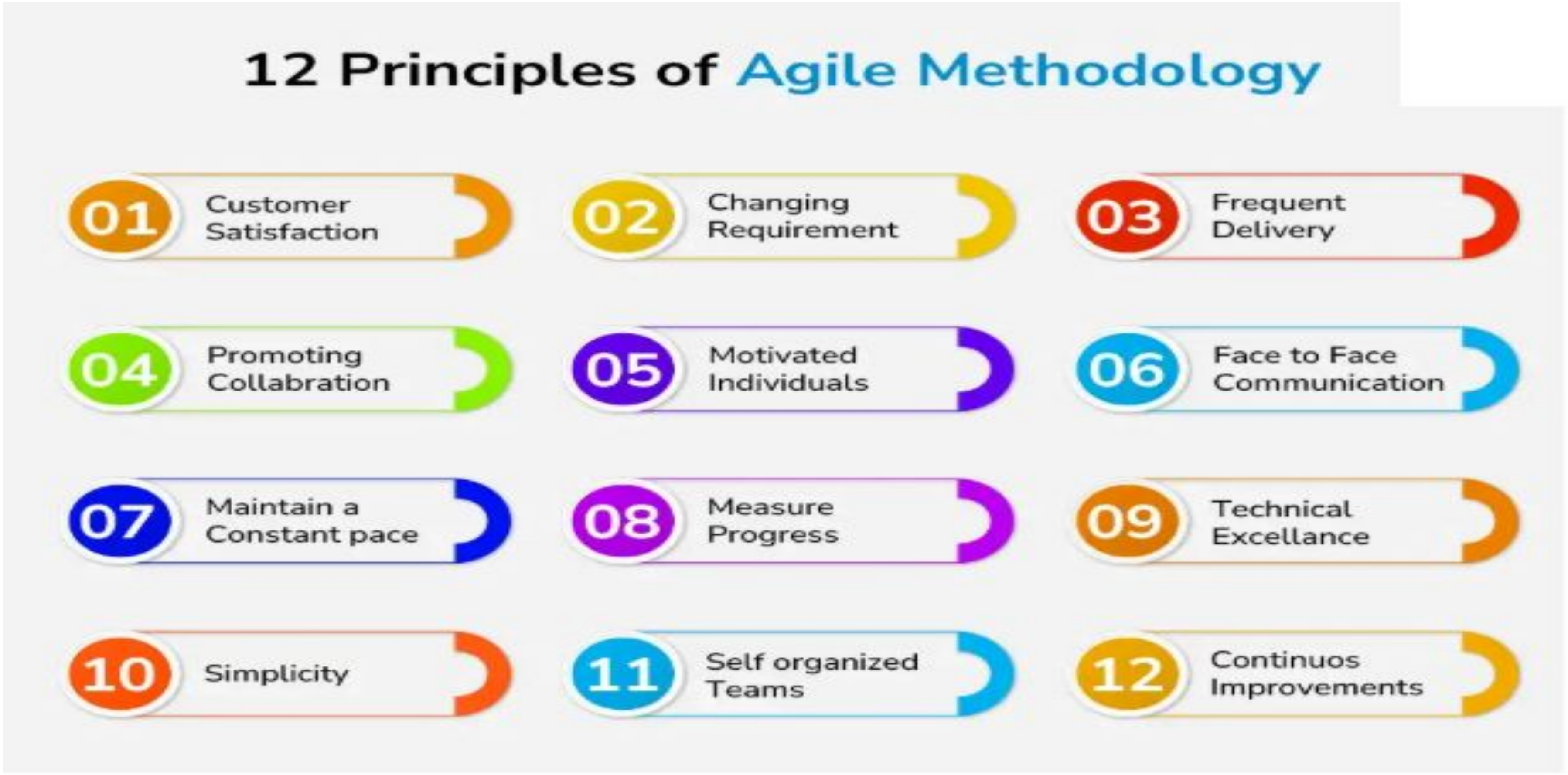
- After each sprint, teams hold a retrospective to identify what went well and what needs improvement.

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

Sessions 9 & 10

- Introduction to DevOps
- DevOps ecosystem
- DevOps phases
- Introduction to containerisation
- Introduction to docker
- Creating docker images using Dockerfile
- Container life cycle

DevOps

Instead of working in silos, both teams collaborate throughout the entire software lifecycle—from planning and coding to testing, deployment, and monitoring.

Challenges in a Regular Application Deployment

- Many a time, the Ops team is unaware of the tech parts required for the app: which version of the framework that has been used, OS versions, dependencies, plugins and many more. Even a change in a config file, they are not aware of it. It used to take days together to resolve these minor issues.

- Integration of multiple config files was a humongous task and would take weeks to resolve them.

- Apps are expected to be created partially, deployed and tested on the current setup instead of creating the whole app and finally testing it. Only during the final deployments, things were figured out and explained to the developers of the expectations which might have altered the actual development.

- The solution to all such problems is : Devs + Ops = DevOps.

- Both the teams are expected to work from the beginning till the end of the Product closure. From sprint planning, collaborative work, resolving deployment issues, config managements and many more. In this case, both the teams are aware of each other's work. The ops team will be with the dev team during the first cycle itself and their dependencies will gradually reduce over the time.

- As both the teams work together hand in hand, they understand the environment on both the ends, resolve the issues quickly instead of waiting till the end, the final deployment will be easier as both the teams are aware of the environment and they know on how to solve the issues if raised.

Introduction to DevOps

- DevOps is a modern approach to software development that combines Development (Dev) and Operations (Ops) to improve the speed, quality, and efficiency of software delivery.
- It focuses on collaboration, automation, and continuous integration and delivery (CI/CD) to deliver applications and services faster and more reliably

Objectives of DevOps:

- Faster software delivery
- Improved collaboration between development and operations teams
- Automation of manual tasks (e.g., testing, deployment)
- Continuous feedback for improvement
- High system stability and performance

Main Components of DevOps:

- Continuous Integration (CI): Developers frequently integrate their code into a shared repository. Automated tests are run to detect bugs early.
- Continuous Delivery (CD): Software is automatically delivered to production or staging environments after successful CI.

- Infrastructure as Code (IaC) : Infrastructure (like servers and networks) is managed using code and automation tools.
- Monitoring and Logging : Tools track the performance of the application and identify any issues in real time.

Stage	**Popular Tools**
Planning & Tracking	Jira, Trello, Azure Boards
Source Control	Git, GitHub, GitLab, Bitbucket
Build & Integration	Jenkins, CircleCI, Travis CI
Testing	Selenium, JUnit, Postman, Cypress
Deployment	Docker, Kubernetes, Ansible, Terraform
Monitoring	Prometheus, Grafana, New Relic, Splunk

DevOps phases

- The DevOps lifecycle is a continuous loop of practices that integrates development and operations teams to deliver high-quality software faster and more reliably.

7 C’s of DevOps:

Continuous Development

- Involves planning and coding.
- Code is written in small, manageable chunks and stored in version control systems like Git.

Continuous Integration (CI)

- Developers frequently merge code into a shared repository.
- Automated builds and tests are triggered to detect issues early.

Continuous Testing

- Automated tests (unit, integration, UI) are run to ensure code quality.
- Tools like Selenium, JUnit, and Cypress are commonly used.

Continuous Deployment/Delivery (CD)

- Code that passes tests is automatically deployed to production or staging environments.
- Ensures faster and more reliable releases.

Continuous Monitoring

- Real-time monitoring of applications and infrastructure.
- Tools like Prometheus, Grafana, and New Relic help detect performance issues and outages.

Continuous Feedback

- Feedback from users and monitoring tools is gathered to improve future iterations.
- Helps teams stay aligned with user needs and expectations.

Continuous Operations

- Focuses on maintaining system uptime and performance.
- Includes automated scaling, failover, and recovery processes.

This lifecycle forms an infinite loop of improvement

Introduction to Containerisation

- Containerisation is the process of lightweight bundling an application together with all its dependencies (libraries, configuration files, binaries) into a single unit called a container.
- This container can run consistently across different environments—whether it's a developer’s laptop, a test server, or a cloud platform.
- Simply put, It is a efficient way to package and run applications—especially useful in modern cloud-native development.

How It Works?

- A Dockerfile defines the configuration of the container.
- The application and its dependencies are built into a container image.
- This image is run as a container on any system with a container runtime.

Why Use Containers?

- Portability : “Write once, run anywhere” becomes a reality.
- Speed : Containers start up in seconds.
- Scalability : Ideal for microservices and cloud-native apps.
- Efficiency : Multiple containers can run on the same machine with minimal overhead.
- Fault Isolation : If one container fails, others remain unaffected.

Popular Tools - Docker, Kubernetes, Podman, CRI-O, containerd

Introduction to Docker

- Docker is an open-source containerization platform that allows developers to package applications along with all their dependencies into containers.
- These containers ensure that the application runs the same across all environments – development, testing, and production.
- Docker simplifies the process of building, shipping, and running applications, making it a key tool in DevOps and cloud-native development.

Why Docker?

- Before Docker, developers faced the common issue of "It works on my machine." Docker solves this by packaging everything an application needs into a lightweight container, ensuring consistency across all systems.

Key Features of Docker:

- Portability - Docker containers can run on any platform that supports Docker – Windows, Linux, or the cloud.

- Lightweight - Containers share the host OS, making them faster and more efficient than virtual machines.

- Isolation - Each container runs independently without interfering with others.

- Fast Deployment - Applications start quickly, enabling faster development and scaling.

Keywords

- Docker Engine - The core runtime that manages containers.
- Dockerfile - A script containing instructions to build a Docker image.
- Docker Image - A snapshot of the application and its environment, used to run containers.
- Docker Container - The running instance of a Docker image.
- Docker Hub - A cloud-based registry to share and download Docker images.

Basic Docker Workflow:

- Write a Dockerfile that defines how to set up the application.
- Build a Docker image using the Dockerfile.
- Run the image as a container.
- Deploy or scale the container on any system or cloud.

Example Use Case:

A web developer can containerize a Node.js application with MongoDB and run it across different environments without worrying about installation issues.

Creating docker images using Dockerfile

- A Dockerfile is a plain text file that contains a set of instructions used to build a Docker image.
- These images are templates for running containers that include the application and all required dependencies.

Docker Commands

- FROM- Specifies the base image (e.g., ubuntu, node)
- RUN - Executes a command (e.g., install packages)
- COPY - Copies files from host to the image
- WORKDIR - Sets the working directory for subsequent commands
- CMD - Defines the default command to run in the container
- EXPOSE - Indicates the port on which the container listens

Steps to Build and Run an Image

Steps for installing a Java App as a container inside a Docker

1. Create a folder called javaApp
2. Develop the requried code for the javaApp. create a Sample class and do a hello world program.
3. Build the App and test it.
4. U must create the Dockerfile, a text file with no extensions. Provide the appropriate instructions

```
Openjdk:11

WORKDIR /var/www/java

COPY . /var/www/java/

RUN javac SampleFile.java

CMD exec java SampleFile
```

- U must restart the machine, then U can download the MSI file of the Docker from
- Install the Docker Desktop from the link https://www.docker.com/products/docker-desktop/
- If required, U can add Path environment variable for executing Docker commands from the cmd.
- Docker starts immediately after installation, however U can change the settings to start on request instead of Auto start.
- If the Cmd is already opened, U may have to restart it to execute the docker commands.

Using MongoDb from the Docker

Run the following commands in the order

```
docker pull mongodb/mongodb-community-server

docker run --name mongo -d mongodb/mongodb-community-server:latest

docker container ls

docker exec -it mongo mongosh

show dbs
```

Run the Docker commands to create the image and execute the Container.

```
docker build -t java-app .

docker run java-app
```

Options:

- -t => to run the app in a virtual terminal so that U can see the results.
- -i => Use this option if U want the app run in interactive mode, when U expect User inputs from the Console Window.

Lab

- Install and configure docker
- Create docker image using Dockerfile
- Start docker container
- Connect to docker container
- Copy the website code to the container
- Use docker management commands to
  o List the images
  o List the containers
  o Start and stop container
  o Remove container and image

Session 11

- Introduction to YAML
- Introduction to Docker Swarm and Docker Stack
- Introduction to Kubernetes
- Creating Kubernetes cluster
- Creating service in Kubernetes
- Deploying an application using dashboard

Introduction to YAML

- YAML (YAML Ain't Markup Language) is a human-readable data serialization format often used for configuration files in DevOps tools like Docker, Kubernetes, and Ansible.

Features:

- Uses indentation (spaces, not tabs) to represent structure
- Supports key-value pairs, lists, and nested data
- File extension: `.yaml` or `.yml`

Introduction to Docker Swarm and Docker Stack

- Docker Swarm is Docker's native clustering and orchestration tool.
- It allows you to manage a group of Docker engines as a single virtual system.
    - docker swarm init : Initializes a Swarm
    - docker service create : Deploys services in the Swarm
- Docker Stack Docker Stack is used to deploy multi-service applications defined in a docker-compose.yml file into a Swarm.

Introduction to Kubernetes

- It is a container management tool developed by Google. its main purpose is helping in managing the containerized apps on various platforms of cloud. It can also reside in Virtual servers and local servers. It is said to be one of the most popular containerization management tools.
- It is purely cloud based and comes with host of communication and automation tools that are used to maintain and manage large scale containers as one unit.
- K8s maintains clusters for managing the services. These services are grouped into nodes and pods. A service can have one or more nodes and each node can have one or more PODs. Each POD represents an independent container of microservices.
- K8s maintains these services using multiple clustors. When an App is required to be loaded, it will be loaded into Primary Clustor. When, on for some reasons, the primary clustors fail to load, it immediately invokes a secondary clustor(backup) and the repo of the secondary clustor will be provided and the app will not break down.

How K8s work?

- It a linux based Environment that shares lots of resources required to manage complex Apps. It is primarily used for distributed computing apps where the K8s abstract the underlying infrastructure and hardware resources and offers std and consistent UI that a DevOps Engineer can monitor from a common place.
- The UI tool looks simple yet allowing to perform complex operations.
- It works similar to Jenkins where one can monitor multiple apps, clusters and allocate the resources required for each of the application.
- The DevOps person can determine the amount of resources that each app needs and allocate them by either scaling up or scaling down the resources for the app to run smoothly.

Microservices:

1. They are small scale apps that are created while breaking down large app into smaller modular units which are designed to work independently of one another.
2. With the new development methodologies, where the final product is never understood and only small updations of app happen over short intervals of time, microservices are the way to go.
3. UR Service is hosted insider the serverless environment registered under a service broker who publishes your services and allows customer to discover the services based on their search engines and the service providers they are having business with.
4. Microservices can be developed under various technologies like Java-Springboot, .NET CORE and .NET WEB API and other open source projects like NODEJS-EXPRESS , NESTJS and many more.
5. Our example will be on .NET CORE to develop an ASP.NET CORE WEB API Project that will connect a SQL server database. We will have 2 Docker containers that will have our App in one Docker image and the SQL server in another docker image. They both will be integrated using a language called YAML. We use Docker Compose tool to orchestrate the interaction between the .NET CORE and SQL server database.
6. U can try creating a REACT App that consumes this REST API. U will additional middleware like CORS and other DI tools.

How to create Microservices

- VS 2022 should be selected. Create a ASP.NET CORE Web API project
- Better go for CODE FIRST Approach to create the data classes. Include the required EF Tools
- Implement the DBContext and provide the required DI feature into Program.cs
- If connecting to SQL server, we will create an YAML file for downloading image of SQL server in the Docker compose.
- Expose the service to be called by any Application.

Creating a Web API microservice:

Implementation of the service:

1. Create a new ASP.NET Core Web API project WebApiSqlServerDockerDemo in Visual Studio 2022 using .NET 7.0 and make sure you enable the Docker support while you are creating the project.
2. Open the NuGet package manager and search and install the following packages in your project.
    - Microsoft.EntityFrameworkCore.SqlServer
    - Microsoft.EntityFrameworkCore.Design
    - Microsoft.EntityFrameworkCore.Tools
3. Create a Models folder in the project root folder and create the following Product entity in that folder with the code shared.
4. create a folder called Data in our project and create a new class called OnlineShopDbContext in this folder with the code shared.
5. Define our database connection string and we can save the connection string in the appsettings.json file.
6. Register the SQL Server database provider in Program.cs file using the UseSqlServer method. The UseSqlServer method requires a database connection string and we can read and pass this information using the GetConnectionString method.

7. Implement the controller class.
8. Add Docker compose support in Visual Studio, right-click on the Web API project in the solution explorer and choose and choose Add > Container Orchestrator Support… option from the menu.
9. Follow the steps to choose Docker Compose and Linux machine
10. Build the Application with the Docker Compose as Start up to allow the containers created in the Docker
11. Run the Application.
12. U can configure the Sql Server from the SSMS

Issues with K8s:

- It needs a heavy infrastructure to showcase any application
- The complete pipeline is done by a team of testers, Devops Engineers and QAT members.
- It is a collaborative work to make UR services hosted in K8s Server. It is not user friendly like Jenkins.
- But there are many 3rd party UI tools that can manage this infrastructure.

Lab :

- Configure Kubernetes
- Configure Kubernetes Dashboard
- Setup a Kubernetes cluster
- Access application using Kubernetes service
- Deploy the website using Dashboard