Software Development Methodologies

Contents

Waterfall, iterative

Software Engineering

- ✓ SDLC
- ✓ Agile
- types, methods, levels — Testing
- DevOps lifecycle : tools — DevOps

feature branching

version control system → git

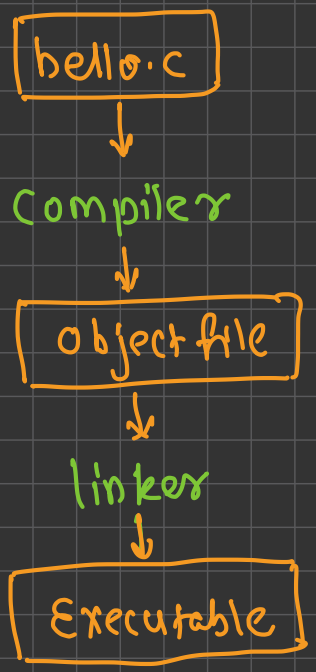
- SCM
- Docker, scalability → HA apps — Containerization
- Kubernetes
- container management : Docker swarm — Container Orchestration
- Jenkins, TravisCI ... — CI/CD Pipeline

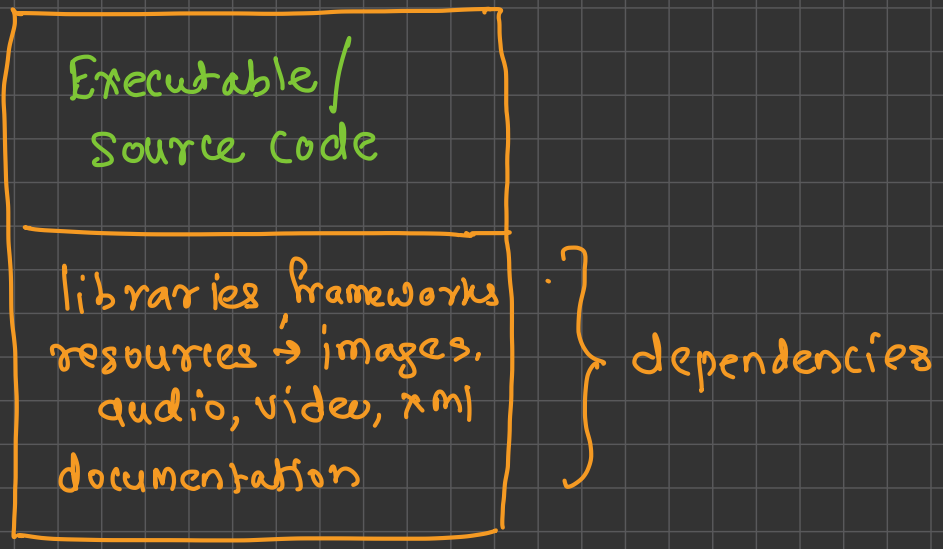
cloud → AWS

About Instructor

- With **18+ years of crafting technology solutions**, I bring a blend of innovation, leadership, and hands-on expertise
- As the **Associate Technical Director at Sunbeam** and a passionate **Freelance Developer**, I've explored diverse domains, solving complex problems with modern tools and technologies.
- I've had the privilege of developing **many mobile applications** that power experiences on **iOS** and **Android** platforms and crafting dynamic websites with **PHP**, **MEAN**, **and MERN stacks**
- My journey has been fuelled by a deep love for programming languages like **C, C++, Python, JavaScript, TypeScript, Go, Swift, Kotlin and PHP**
- My DevOps journey includes building **CI/CD pipelines using Jenkins, GitHub Actions & ArgoCD**, automating infrastructure with tools like **Terraform & Ansible**, leveraging cloud platforms like **AWS** to create robust, scalable systems and containerizing applications using **Docker** and orchestrating them using **Kubernetes and Docker Swarm**

Software Engineering





Software → package

$$\text{Software} = \frac{\text{Executable}}{\text{Source code}} + \text{dependencies}$$

Software product = Software developed for solving a particular problem, → photoshop, office

Introduction

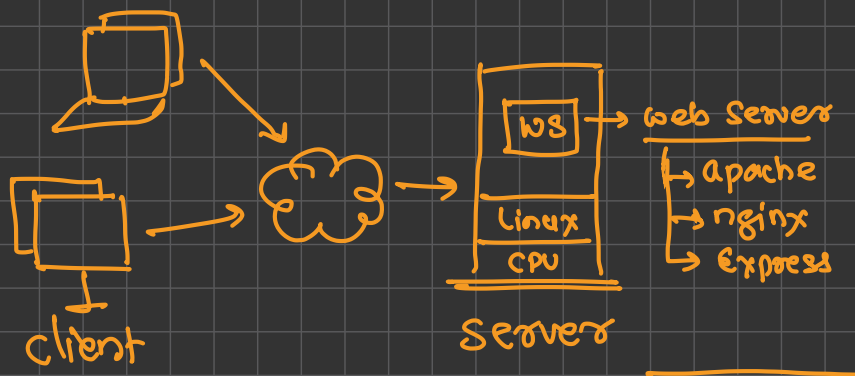
- Software Engineering is the systematic application of engineering principles to the design, development, testing, deployment, and maintenance of software systems.
- In simple terms:
  - It's about building high-quality, reliable, and maintainable software in a structured, disciplined way — not just coding!
- **Goals of Software Engineering**
  - Build software that works correctly and efficiently
  - Ensure it's maintainable, scalable, and secure
  - Deliver software on time and within budget
  - Satisfy user needs and business requirements.

→ requirement analysis
→ testing
→ stages
→ discipline

Scaling → making app available to many users

① vertical scaling

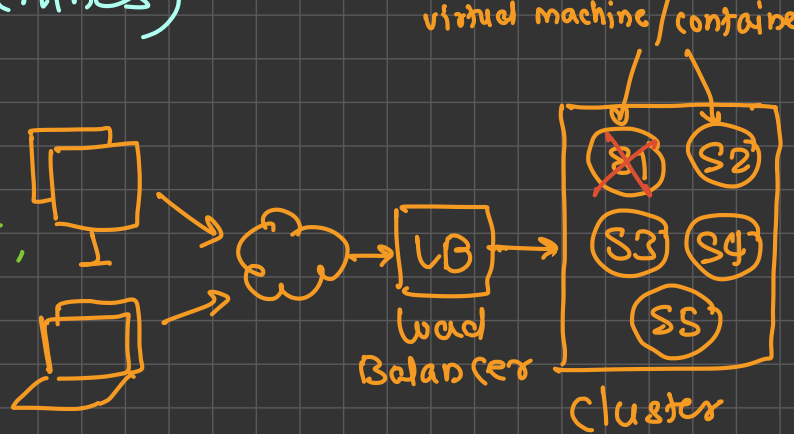
→ increasing / reducing resources on server
→ used for static scaling → known load
→ simple, no code change, cheaper than horizontal
→ downtime, limitation on hardware,
single point of failure (if machine goes down, the whole app goes down)



1x CPU → 30 threads
4x CPU = 30 x 4 = 120 Threads

② horizontal scaling (↑ replicas of app)

→ adding / removing servers: (physical / virtual) machines, containers
→ uses cluster of servers (group of machines)
→ highly available app → app won't go down
→ HA app, dynamic load handling, no downtime,
no limit on no of servers to be added to the cluster,
no single point of failure
→ costlier than vertical, complex, code changes required



Principles of Software Engineering

- **Modularity:** Divide system into independent modules. → node modules, packages
- **Abstraction:** Focus on what a system does, not how. → interface designing
- **Encapsulation:** Keep data and functions together.
- **Scalability:** Software should handle growth. → vertical or horizontal
- **Reusability:** Reuse existing code to save effort.
- **Maintainability:** Code should be easy to modify and fix. → enhancements

Characteristics of good software

- **Operational** → functionality point of view
  - **Budget:** Software should be developed within the budget ← time, financial
  - **Usability:** Software should be usable by the end user → should not be complex, simple workflow, localization → app is available in local language
  - **Efficiently:** Software should efficiently use the storage and space → resources
  - **Correctness:** Software should provide all the functionalities correctly without having any bug/issue
  - **Functionality:** Software should meet all the requirements of the user
  - **Dependability:** Software should contain all the dependencies in its package → (libraries, frameworks, resources)
  - **Security:** Software should keep the data safe from any external threat → CIA triad
  - **Safety:** Software should not be hazardous or harmful to the envrionment
- **Transitional** → supporting multiple platforms / environments
  - **Portability:** If the software can perform the same operations on different environments and platforms (→ OS), that shows its Portability
  - **Interoperability:** It is the ability of the software to use the information transparently
  - **Reusability:** If on doing slight modifications to the code of the software, we can use it for a different purpose, then it is reusable
  - **Adaptability:** It is an ability to adapt the new changes in the environment
- **Maintenance**
  - **Maintainability:** The software should be easy to maintain by any user
  - **Flexibility:** The software should be flexible to any changes made to it
  - **Extensibility:** There should not be any problem with the software on increasing the number of functions performed by it (→ feature Enhancement)
  - **Testability:** It should be easy to test the software → Test Driven Development (TDD) → unit testing
  - **Modularity:** A software is of high modularity if it can be divided into separate independent parts and can be modified and tested separately

security

CIA triad

→ Confidentiality → confidential (password, SSN) data must be confidential / sensitive
  ↳ encryption → bcrypt, crypto-JS

→ integrity → what goes in is what goes out

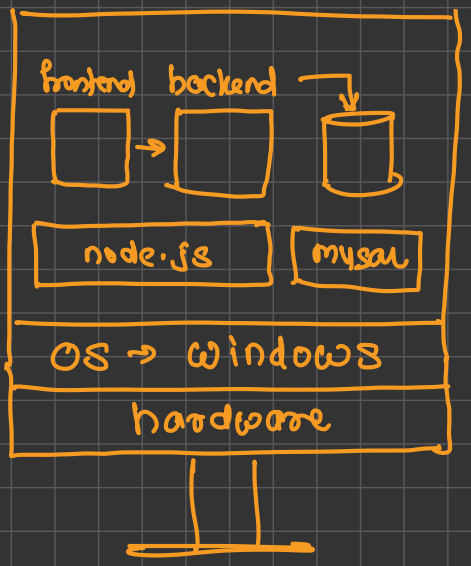
→ Availability → the app must be available all the time to valid users
  ↳ scaling → horizontal type

→ Authentication → checking if user is valid
  ↳ login and registration
  ↳ MFA - multi factor Authentication → OTP

→ Authorization → checking if user has enough permissions
  ↳ JWT tokens
  ↳ RBAC → Role based Access Control

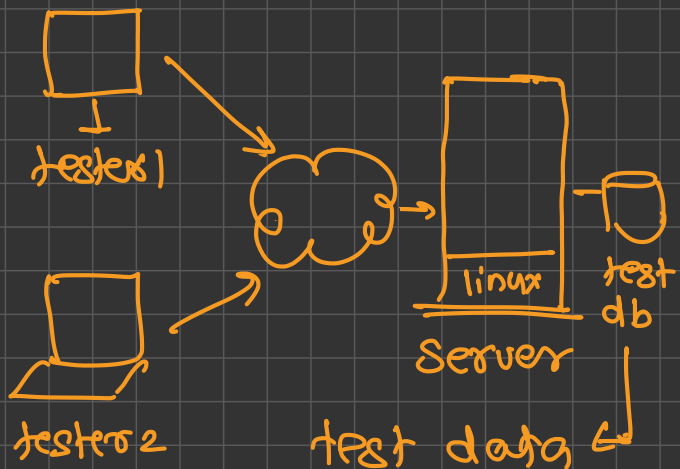
Developer environment

→ developers → development



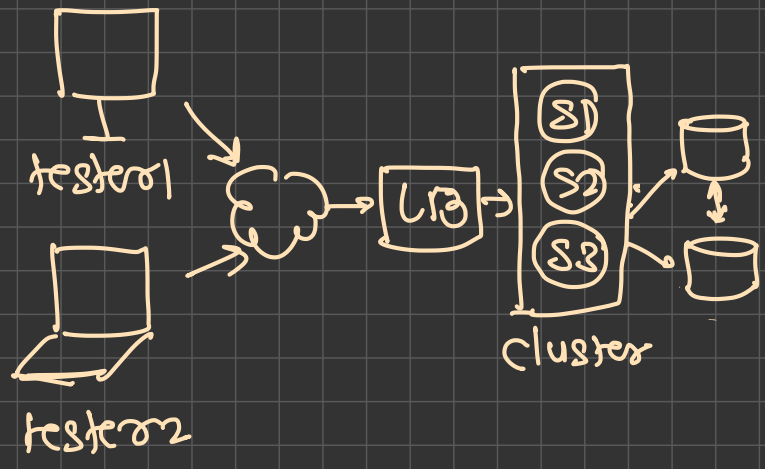
staging / testing environment

→ testers for functional testing

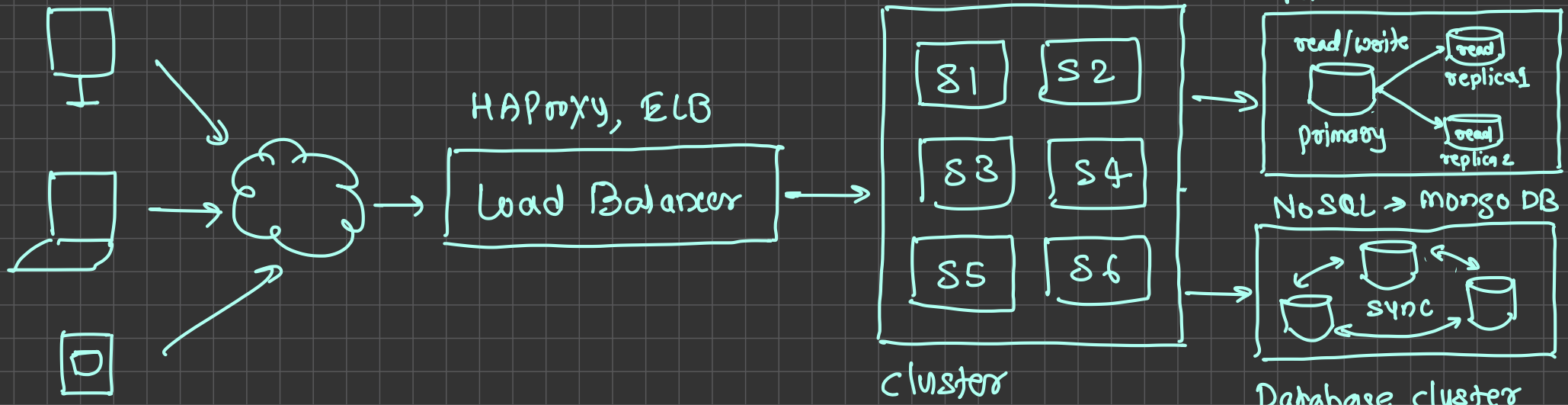


pre-production / UAT environment

→ testers for functional & non-functional testing



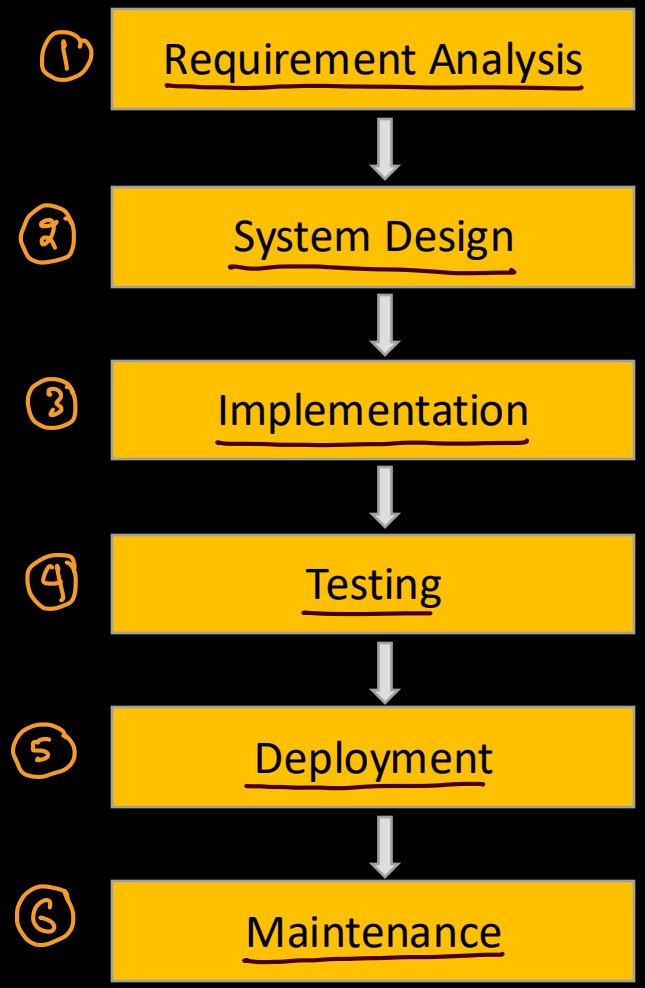
production / live environment → used by end users for executing / running app



Software Development Life Cycle

Overview

- Also called as Software Development Process (SDP) → SDLC
- Is a well-defined, structured sequence of stages in software engineering to develop the intended software product
- Is a framework defining tasks performed at each step in the software development process
- Aims to produce a high-quality software that
  - meets or exceeds customer expectations
  - reaches completion within times and cost estimates
- Consists of detailed plan (stages) of describing how to develop, test, deploy and maintain the software product

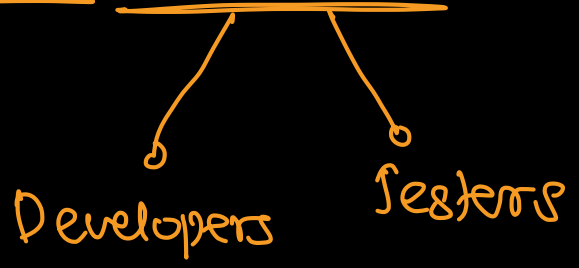


Requirement Analysis → understanding → Requirement gathering

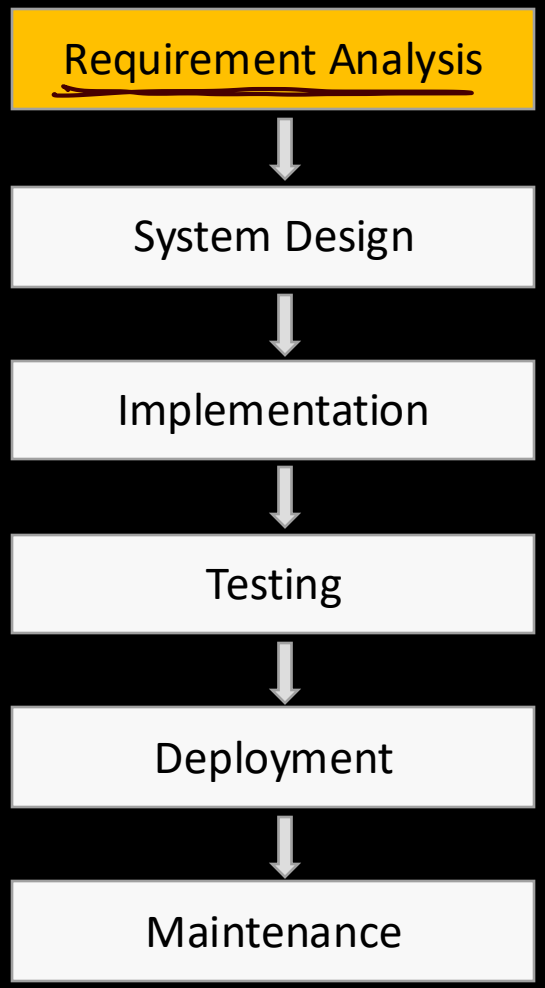
- **Goal:** Understand what needs to be built
- **Activities:**
  - Gather requirements from stakeholders, clients, or end users
  - Create a Software Requirement Specification (SRS) document
  - Identify technical and business feasibility
- **Output:** SRS document

→ interested in the app → owner, VC, investors

→ Contains all requirements to complete the software development

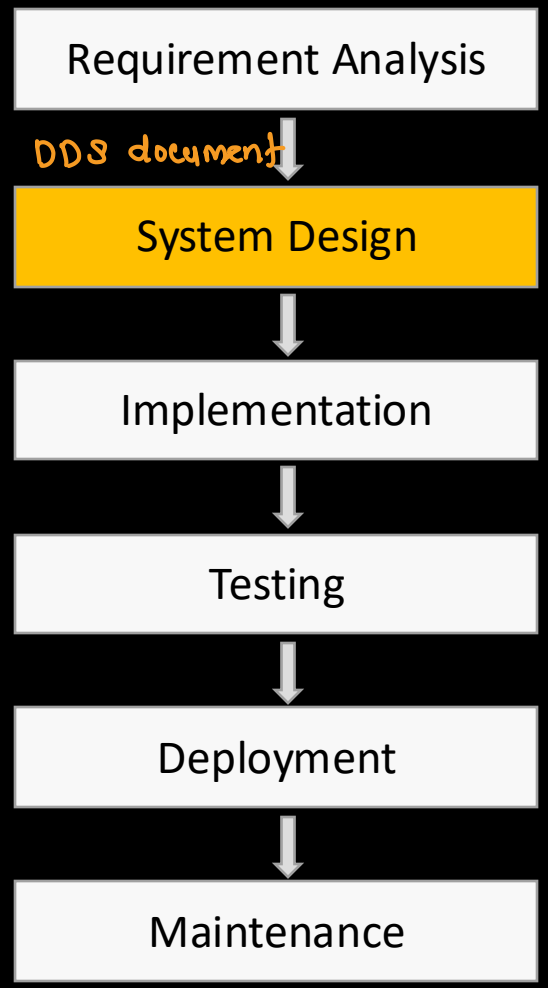


SRS document



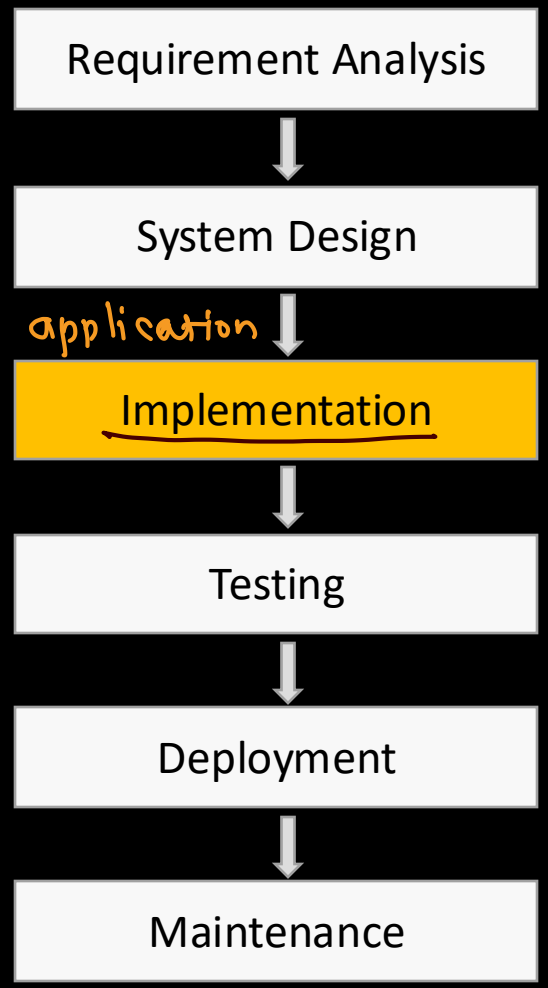
System Design

- **Goal:** Plan how the software will be built
- **Activities:**
  - Define architecture, data flow, and database schema
  - Create UI/UX wireframes
  - Select technologies (frontend, backend, database, etc.)
- **Output:** Design Documents (HLD, LLD — High/Low-Level Design)
  - ↳ Design Document Specification (DDS)



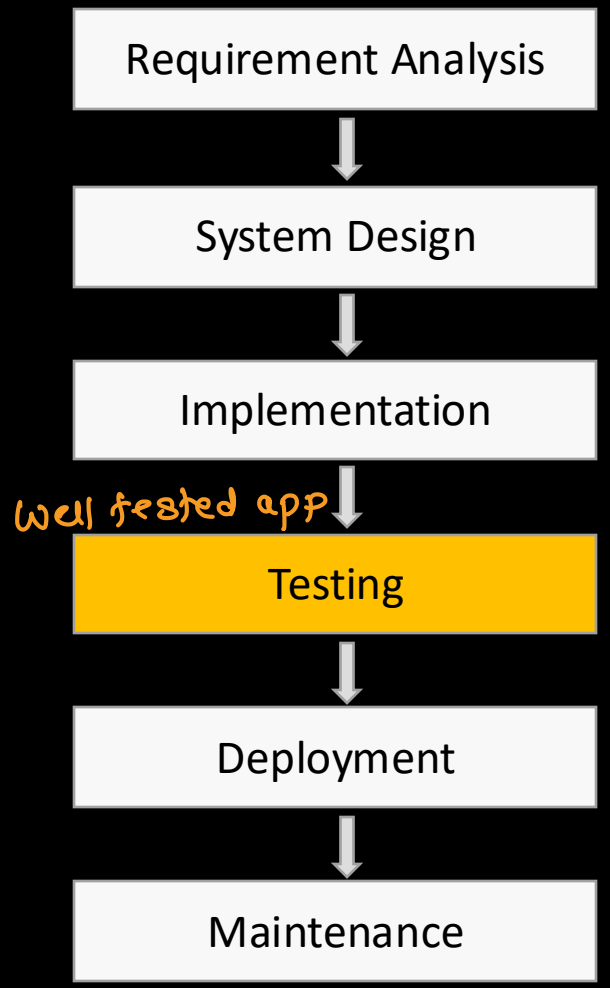
Implementation → programming / coding / Development

- **Goal:** Convert design into working code
- **Activities:**
  - Developers write code following coding standards
  - Use version control (Git) for collaboration
  - Conduct unit testing for each module → unit testing → function
- **Output:** Working source code
  - ↳ System



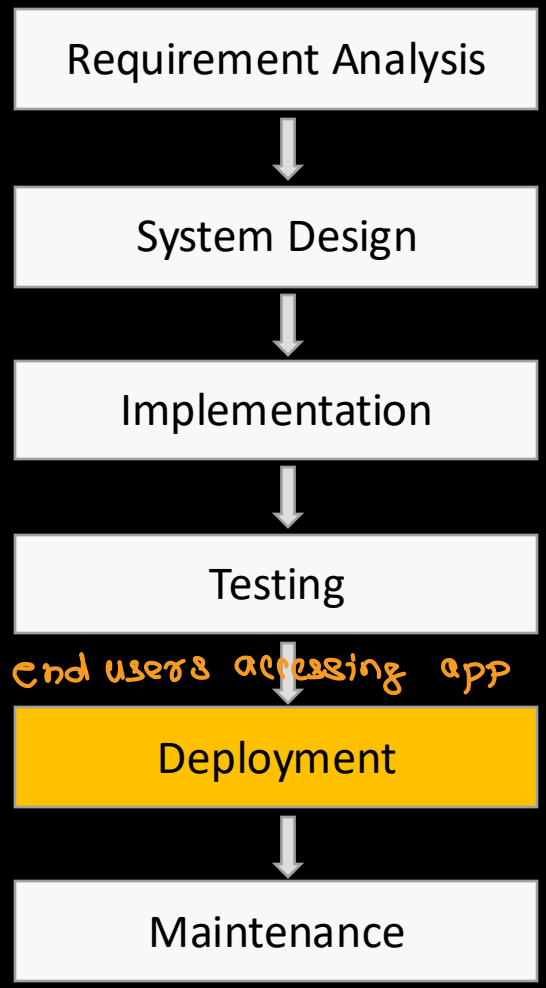
Testing → testers

- **Goal:** Ensure software works correctly and meets requirements from SRS
- **Activities:**
  - Perform unit, integration, system, and user acceptance testing (UAT)
  - Identify and fix bugs
  - Validate performance, security, and compatibility
- **Output:** Tested and verified application



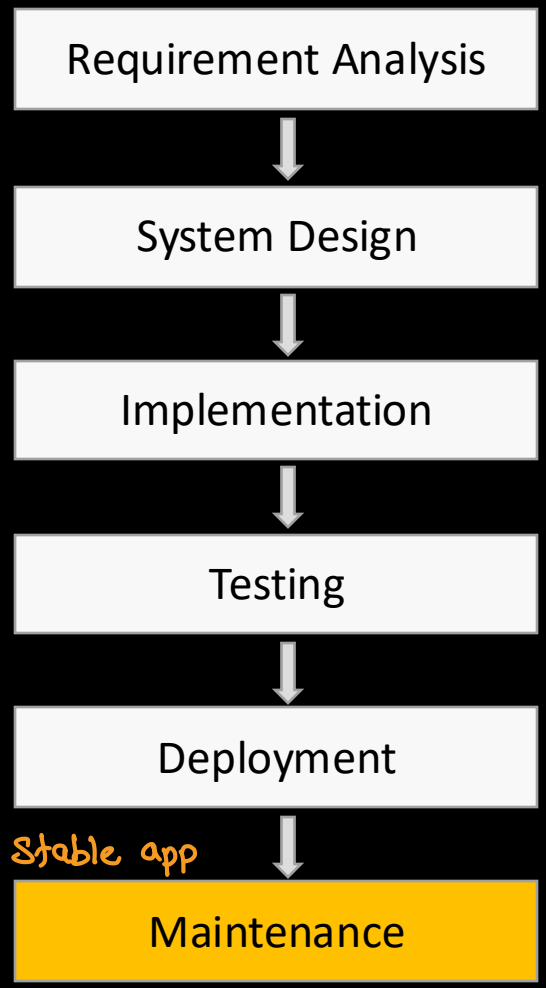
Deployment → moving app from one to another env

- **Goal:** Deliver the product to the end users or production environment
- **Activities:** → Jenkins
  - Deploy software using CI/CD pipelines or manual methods
  - Conduct smoke testing in production → quick testing
  - Monitor for early issues
- **Output:** Software available to users



Maintenance

- **Goal:** Keep the software running smoothly after release
- **Activities:**
  - Fix bugs reported by users (end)
  - Update software with new features or security patches (→ enhancement)
  - Improve performance → performance testing
- **Output:** Updated and stable software



SDLC Models → framework following the idea/implementation of SDLC

Model	Description	When to Use
Waterfall	Linear and sequential — each phase follows the previous one.	Small, well-defined projects.
Agile	Iterative and incremental — work in short sprints with continuous feedback.	Dynamic projects with evolving requirements.
Spiral	Combines design and prototyping with risk analysis.	Large, complex, and high-risk projects.
V-Model	Extension of Waterfall with testing after every phase.	When strict validation is required.
Iterative	Develop a version, test, improve, and repeat.	When gradual improvement is needed.
DevOps	Continuous integration, delivery, and monitoring.	For modern, cloud-based environments needing fast releases.

Benefits of SDLC

stages

- Ensures **structured development** and **predictability**
- Improves **project planning** and **resource management**
- Reduces **project risks** and **development costs**
- Delivers **high-quality** software that meets requirements

Agile Methodologies

Agile Development

- **Agile Development** is a **flexible**, **iterative approach** to software development that focuses on **continuous delivery**, **collaboration**, and **customer feedback**
- Instead of building the entire product in one go (like in the **Waterfall model**), Agile breaks the project into **small parts** called **iterations** or **sprints** — each delivering a **working piece of software**
- Agile = "Build small, deliver fast, get feedback, and improve continuously."
- The Agile Manifesto (Core Philosophy)
  - In 2001, 17 software developers wrote the **Agile Manifesto**, which defines **4 key values** and **12 principles** for better software development

Traditional Focus	Agile Focus
Processes and tools	**Individuals and interactions**
Comprehensive documentation	**Working software**
Contract negotiation	**Customer collaboration**
Following a plan	**Responding to change**

product backlog → collection of all the stories

* E-commerce
  - user module → * Registration of user ← Epic
  - product module
  - order module
  - AI module
  - Notification module

① → create a table named users
② → create an API - /user/register
③ → create frontend page - /register
④ → connect component to API

← story → action/task

* Authenticate user ← Epic

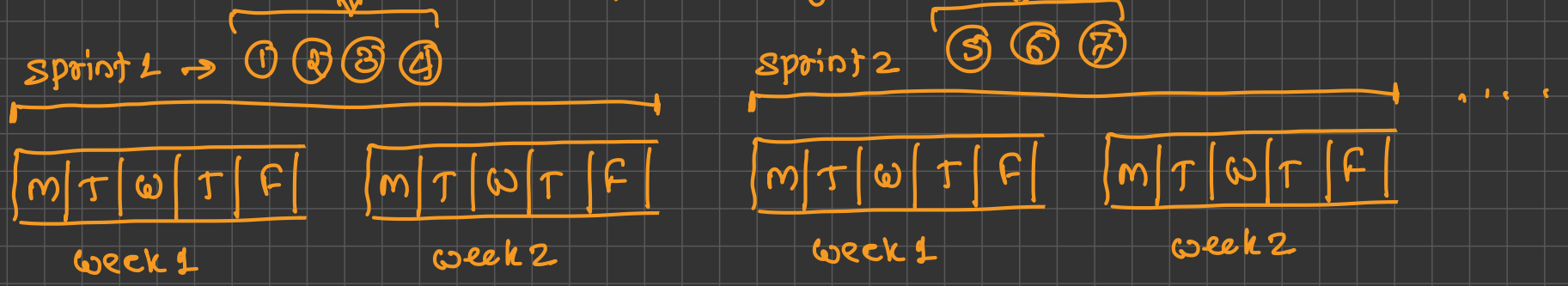
⑤ → create an API - /user/login
⑥ → create frontend page - /login
⑦ → connect component to API

* forgot password ← Epic

. . . .

Collection of stories selected in a sprint

sprint backlog



sprint → 2 weeks
sprint → time boxed event (1 - 4 weeks)

sprint → 2 weeks

Components of Agile Development

Component	Description
Sprint	A short, time-boxed period (1–4 weeks) where a small part of the product is developed.
Product Backlog	A prioritized list of features or user stories that need to be implemented.
Sprint Backlog	Subset of the product backlog selected for a specific sprint.
Daily Stand-up	A short 15-minute daily meeting to track progress and obstacles.
Sprint Review	Meeting at the end of the sprint to demonstrate completed work.
Sprint Retrospective	A reflection session to improve team processes in the next sprint.

↳ sprint backlog

daily scrum

Agile Frameworks → way to develop app using Agile methodologies

Framework	Highlights
Scrum ⁂	Most popular Agile framework; uses sprints, roles, and ceremonies.
Kanban	Visual workflow (board with “To Do → Doing → Done”) focusing on continuous flow.
Extreme Programming (XP)	Emphasizes coding practices like pair programming and continuous integration.
Lean	Focuses on minimizing waste and maximizing value delivery.
SAFe (Scaled Agile Framework)	Extends Agile for large, enterprise-level teams.

What is Scrum ?

→ Roles
→ Artifacts
→ events
→ teams

- **Scrum** is an **Agile framework** used to develop, deliver, and sustain complex software projects
- It emphasizes **team collaboration**, **iterative progress**, **continuous feedback**, and **flexibility**
- Scrum helps teams deliver **working software quickly**, in **small increments**, while continuously improving
- Scrum divides the entire project into **small, manageable chunks** called **Sprints** (typically 2–4 weeks)
- Each sprint results in a **potentially shippable product increment** — something working and usable
- At the end of every sprint:
  - The team reviews progress
  - The product is demonstrated
  - Feedback is gathered
  - Improvements are planned for the next sprint

Scrum Roles

- **Product Owner**
  - Represents the customer or stakeholders
  - Manages the Product Backlog
  - Prioritizes features and defines User Stories
  - Ensures the team builds what provides the most value
- **Scrum Master**
  - Acts as a coach and facilitator for the Scrum team
  - Removes obstacles (impediments)
  - Ensures Scrum principles and ceremonies are followed
  - Shields team from distractions
- **Development Team**
  - Cross-functional members (developers, testers, designers, etc.)
  - Self-organizing — decide how to build the product
  - Delivers a working product increment every sprint

dev team
- developers
- testers

ops team

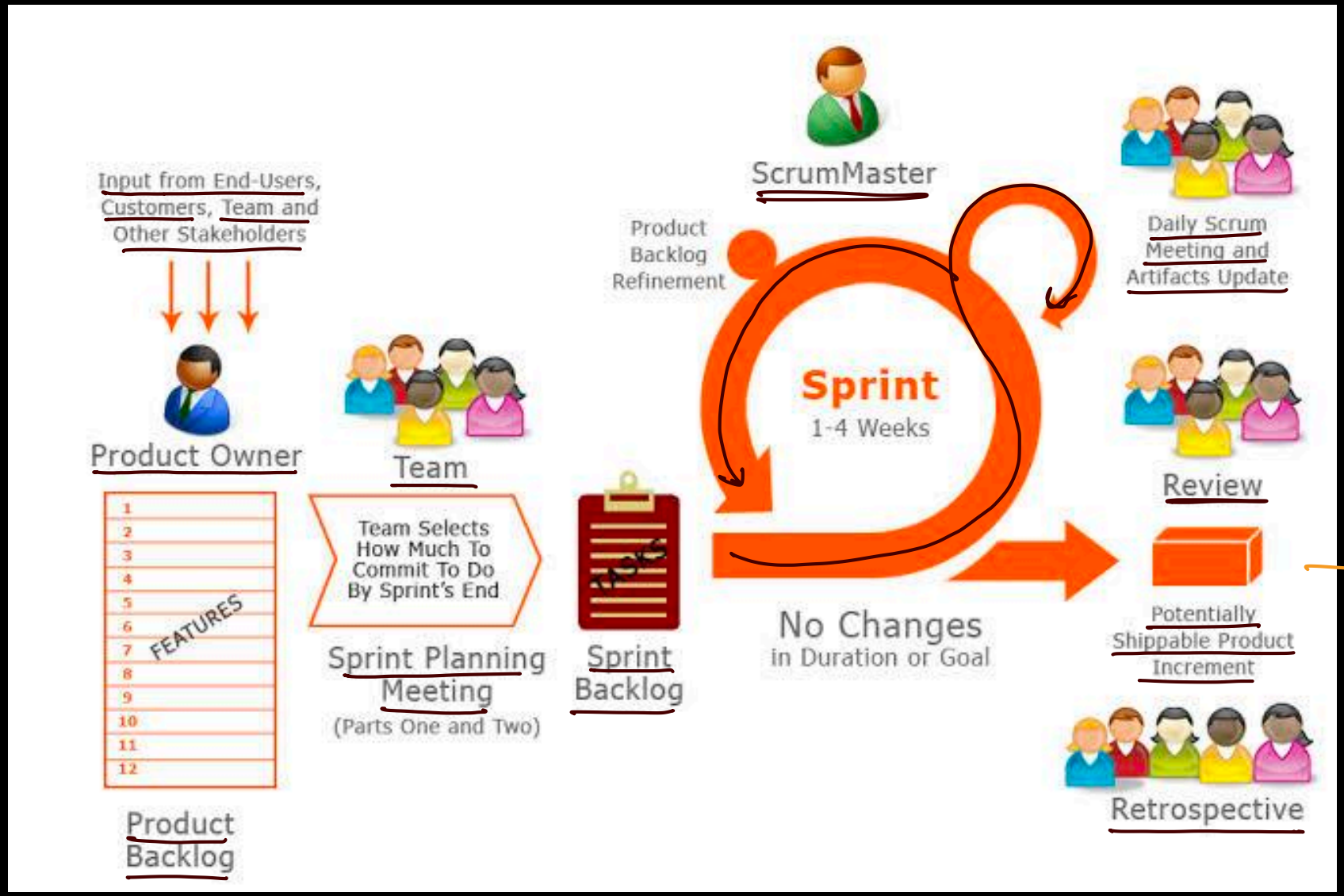
Scrum Artifacts (Key Documents)

- **Product Backlog**
  - A prioritized list of everything needed in the product (features, bugs, improvements)
  - Managed by the Product Owner with the help from scrum master
- **Sprint Backlog**
  - A subset of the Product Backlog selected for the current sprint
  - Includes tasks and a goal → needs to be achieved by sprint
- **Increment**
  - The working product that results from a sprint — it must be usable and meet the "Definition of Done"
- **Definition of Done (DoD)**
  - A shared understanding of what "done" means (e.g., coded, tested, documented, deployed)

Scrum Ceremonies (Events) → meetings

When	Ceremony	Purpose	Duration
before sprint	**Sprint Planning**	Define the **Sprint Goal** and decide **which backlog items** to work on. → sprint backlog	2–4 hours (for 2-week sprint)
Every Day	**Daily Scrum (Stand-up)**	15-min meeting where team members share:1. What they did yesterday2. What they'll do today3. Any blockers	15 minutes daily
after sprint	**Sprint Review**	Demonstrate the **completed work** to stakeholders and collect feedback.	1–2 hours
after Sprint	**Sprint Retrospective**	Discuss **what went well**, **what didn't**, and **how to improve** next sprint.	1–1.5 hours
–	**The Sprint**	The actual 1–4 week **development cycle** where work happens.	Fixed timebox

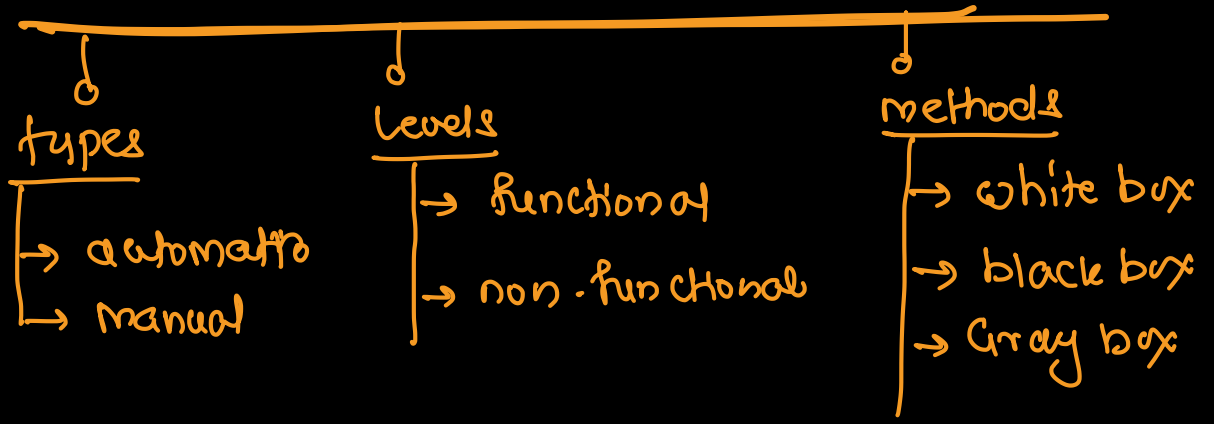
Scrum Workflow



Scrum tools

- Jira – https://www.atlassian.com/software/jira/
- Clarizen – https://www.clarizen.com/
- GitScrum – https://site.gitscrum.com/
- Vivify Scrum – https://www.vivifyscrum.com/
- Yodiz – https://www.yodiz.com/
- ScrumDo – https://www.scrumdo.com/
- Quickscrum – https://www.quickscrum.com/
- Manuscript – https://www.manuscript.com/
- Scrumwise – https://www.scrumwise.com/
- Axosoft – https://www.axosoft.com/

Application Testing



Introduction

developers testers

- **Application Testing** is the **process of verifying and validating** that a software application:
  - Works as intended → SRS
  - Meets business and user requirements → SRS
  - Is free from defects or bugs ≠ gap (missing functionality)
  - Performs well under various conditions → at every stage of SDLC
- In simple terms:
  - Application Testing ensures the **software is reliable, secure, and user-friendly** before it goes live
- **Goals of Application Testing**
  - Detect and fix **bugs** early
  - Ensure the **functionality** meets requirements → SRS
  - Verify **performance, usability, and security**
  - Prevent future defects through **regression tests**
  - Ensure **cross-platform** and **cross-browser compatibility** (for web & mobile apps)

verification
→ static activity
→ conducted by developers
→ while developing app
→ checking code, quality, coverage, unit testing

validation
→ dynamic activity
→ app is already developed
→ performed by tester
→ test cases, execution of test suites

bug
→ defect → not working as expected
→ gap → missing functionality
→ new feature

Phases of Application Testing → STLC

→ manual
→ automated
→ white/ black/ gray box
↳ Selenium/ cypress

Phase	Description
Test Planning	Define test objectives, strategy, tools, scope, and resources.
Test Design	Create **test cases**, **test data**, and define expected outcomes.
Test Environment Setup	Prepare hardware, software, and network needed for testing.
Test Execution	Run test cases and record results. → test case ≤ success / failure
Defect Reporting & Tracking	Log bugs in tools (like Jira, Bugzilla).
Retesting & Regression Testing	Re-test after bug fixes and ensure no new issues appear.
Test Closure	Prepare summary report and finalize testing.

```
function add(x,y){
  return x - y
}
```

test case:
→ pass 20 & 10
→ expect → 30

```
const result = add(20,10)
assert (result == 30)  X
```

test email data
→ amit@test.com → ✓
→ amit@ → X
→ amit test.com → X
→ test → X

testing environments
→ staging
→ pre-production
→ user acceptance testing (UAT) – customer

Application Testing in SDLC

SDLC Phase	Testing Involvement
Requirement	Review for testability.
Design	Verify architecture and data flow.
Implementation	Unit & integration testing.
Testing	Functional, non-functional testing.
Deployment	Smoke testing in production.
Maintenance	Regression & patch testing.

Types of Application Testing

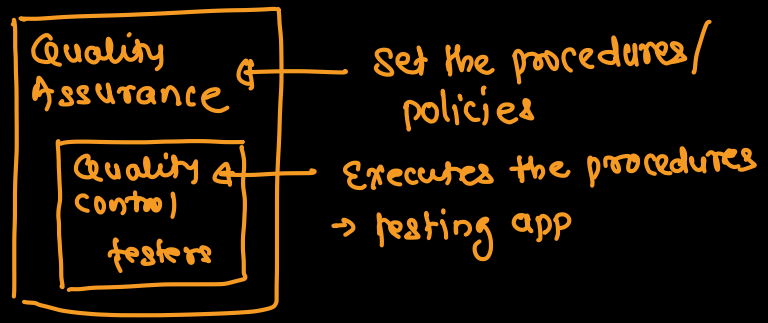
- **Manual Testing** → time taking and error-prone process, cheaper, no or less skills are required
  - Testers execute test cases **without automation tools** (manually)
  - Focus on **exploratory**, **usability**, and **ad-hoc** testing
  - Suitable for **UI validation** and **small projects**
  - Example tools: Excel sheets, TestRail for documentation
- **Automated Testing** → unit integration, performance
  - Test cases are executed using **automation tools** or **scripts**
  - Great for **repetitive**, **regression**, and **performance** tests
  - Saves time and improves accuracy
  - Example tools: Selenium, Cypress, Appium, JUnit, PyTest
  - costlier, requires more skills
  - less error-prone

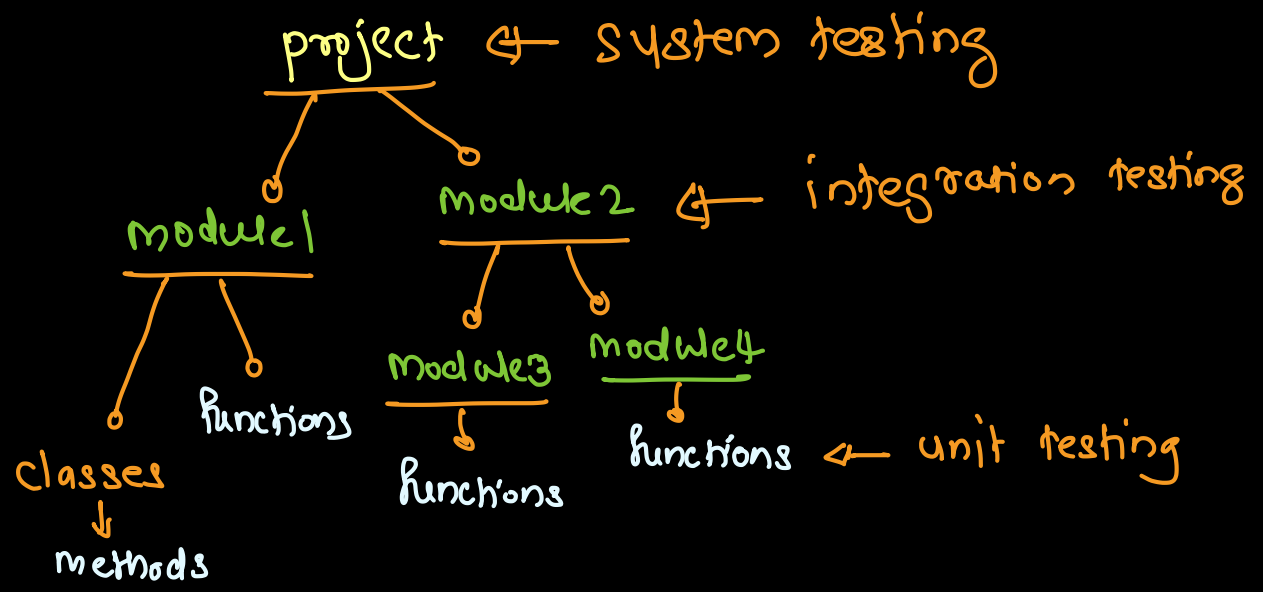
Levels of Application Testing → functional testing levels

Level	Purpose	Performed By	Example Tools
Unit Testing	Test individual modules or functions.	Developers (respective)	JUnit, NUnit, PyTest
Integration Testing	Verify how modules interact together.	Developers/Testers	Postman, JUnit
System Testing	Test the complete application as a whole.	QA Team – testers	Selenium, TestNG
User Acceptance Testing (UAT)	Validate the system with real-world scenarios.	customers End Users / Clients	Manual / UAT tools

→ checking functionality from SRS

→ unit = function





Types of Testing: Functional (system testing) → QA

Type	Description
Smoke Testing	Quick test to ensure basic functions work (build verification).
Sanity Testing	Check new functionality or bug fixes.
Regression Testing	Ensure new changes don't break existing features.
Interface Testing	Validate communication between systems or APIs.
System Testing (E2E testing)	End-to-end testing of the entire system.
User Acceptance Testing (UAT)	Done by end-users to validate business requirements.

Types of Testing: Non-Functional → QA team → validation

tests are performed on system as a whole

Type	Description
Performance Testing	Measure speed, scalability, and stability.
Load Testing	Check system performance under expected load.
Stress Testing	Push beyond limits to see where it breaks. → abnormal condition testing
Security Testing	Ensure data protection and access control. → CIA triad, VAPT
Usability Testing	Verify ease of use and user experience.
Compatibility Testing	Ensure app works on different OS, browsers, and devices.
Accessibility Testing	Ensure app is usable by people with disabilities.

Testing Methods

code review

Testing Method	Tester's Knowledge	Focus Area	Performed By	SDLC Phase
White Box Testing	Full knowledge of internal code, logic, and structure , SRS & DDS	Internal workings (logic, code, paths) → algorithms, practices	Developers	Implementation/ Unit testing
Black Box Testing	No knowledge of internal code	non-functional testing Functionality and output	Testers / QA	Testing/ UAT
Gray Box Testing	Partial knowledge of internal structure	Combination of internal and external	Developers & Testers	Testing/Integration

Common Tools for Application Testing

Category	Popular Tools
Functional Testing	Selenium, Cypress, Playwright, TestNG
API Testing	Postman, SoapUI, Rest Assured
Performance Testing	JMeter, Gatling, LoadRunner
Mobile Testing	Appium, Espresso, XCUITest
Security Testing	OWASP ZAP, Burp Suite
Test Management	Jira, TestRail, Zephyr
CI/CD Integration	Jenkins, GitHub Actions

→ UI testing

Unit testing tools

- → Java → JUnit
- → C# → NUnit
- → python – Pyunit, Pytest
- → JS/TS → Jasmin/Jest

Benefits of Application Testing

- Ensures software quality and reliability
- Detects defects early, reducing costs
- Enhances user satisfaction
- Improves security and performance
- Supports continuous delivery (CI/CD) pipelines

Challenges in Application Testing

- Handling frequent code changes in Agile
- Testing across many browsers/devices
- Managing complex test data
- Balancing speed vs. depth in automated tests