

Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC)

A systematic, phased approach to building high-quality software.

1. Overview

Phase	Primary Goal	Typical Deliverables
1. Planning & Feasibility	Define the problem, assess viability, allocate resources	Project charter, feasibility study, high-level schedule, budget
2. Requirements Analysis	Capture what the system must do from stakeholders	Requirements Specification (functional & non-functional), use-case diagrams, user stories
3. Design	Translate requirements into a blueprint	Architecture diagram, data model, UI mock-ups, component design documents
4. Implementation (Coding)	Build the software according to the design	Source code, unit tests, build scripts
5. Testing & Verification	Validate that the product meets requirements and is defect-free	Test plans, test cases, test reports, defect logs
6. Deployment & Installation	Deliver the product to the target environment	Release notes, deployment scripts, configuration files
7. Maintenance & Support	Keep the system operational, evolve it over time	Patch releases, bug-fixes, enhancements, documentation updates

Note: The SDLC can be executed as a linear “waterfall” sequence, an iterative/incremental cycle, or a hybrid (e.g., V-model, Agile-based). The core phases remain the same; only the flow and feedback loops differ.

2. Detailed Phase Breakdown

2.1 Planning & Feasibility

- **Stakeholder identification** – sponsors, end-users, regulators.

Software Development Life Cycle (SDLC)

- **Scope definition** – boundaries, high-level objectives.
- **Feasibility analysis** –
 - *Technical*: available tech, expertise.
 - *Economic*: cost-benefit, ROI.
 - *Operational*: impact on business processes.
- **Risk assessment** – identify, rank, and propose mitigation.

2.2 Requirements Analysis

- **Elicitation techniques** – interviews, workshops, questionnaires, observation.
- **Documentation** –
 - *Functional requirements*: “The system shall ...”
 - *Non-functional requirements*: performance, security, usability, reliability.
- **Validation** – walkthroughs, prototypes, stakeholder sign-off.

2.3 Design

- **Architectural design** – choose patterns (e.g., MVC, microservices), define layers, decide on technology stack.
- **Detailed design** – class diagrams, sequence diagrams, API contracts, database schema.
- **User-interface design** – wireframes, style guides, accessibility considerations.

2.4 Implementation (Coding)

- **Coding standards** – naming conventions, code style, documentation guidelines.
- **Version control** – Git branching model (feature, develop, release, hotfix).
- **Continuous Integration (CI)** – automated builds, static analysis, unit test execution.

2.5 Testing & Verification

Test Level	Purpose	Typical Tools
Unit	Verify individual components	JUnit, NUnit, pytest
Integration	Check interaction between modules	Postman, SOAP UI
System	Validate complete, integrated system	Selenium, TestComplete

Software Development Life Cycle (SDLC)

Test Level	Purpose	Typical Tools
Acceptance	Confirm business requirements are met	Cucumber, User Acceptance Test (UAT) scripts
Performance / Load	Measure scalability, response times	JMeter, LoadRunner
Security	Identify vulnerabilities	OWASP ZAP, Burp Suite