Software Development Life Cycle (SDLC) Cheat Sheet

A concise reference guide to the Software Development Life Cycle (SDLC), covering various models, phases, and best practices. Ideal for developers, project managers, and anyone involved in software creation.

SDLC Models Overview

Waterfall Model

Description: Linear sequential approach. Each phase must be completed before the next begins.
Best Use: Well-defined requirements, stable technology, and no ambiguous requirements.

Phases: Requirements, Design, Implementation, Testing, Deployment, Maintenance.

Advantages: Simple to understand and implement. Well-suited for projects with clear requirements.

Disadvantages: Inflexible, high risk of late changes, not suitable for complex or evolving projects.

Agile Model

Description: Iterative and incremental approach. Focuses on flexibility and customer collaboration.
Best Use: Projects with evolving requirements and a need for rapid development.

Key Principles: Iterative development, continuous feedback, adaptive planning, self-organizing teams.

Advantages: Flexible, adaptable, high customer satisfaction, suitable for complex projects.

Disadvantages: Requires high customer involvement, can lead to scope creep, documentation can be challenging.

Spiral Model

Description: Risk-driven process model. Combines elements of waterfall and iterative models.
Best Use: High-risk projects with significant uncertainties.

Phases: Planning, Risk Analysis, Engineering, Evaluation.

Advantages: High amount of risk analysis, good for large and complex projects.

Disadvantages: Can be expensive, risk analysis requires expertise, not suitable for small projects.

SDLC Phases Explained

Requirements Gathering

Purpose: Define the scope and objectives of the project.
Activities: Elicit requirements from stakeholders, document user stories, create use cases.

Techniques: Interviews, surveys, brainstorming, prototyping.

Deliverables: Requirements specification document, user stories, use case diagrams.

Best Practices: Involve all stakeholders, prioritize requirements, ensure clarity and completeness.

Design Phase

Purpose: Plan the architecture and structure of the software.
Activities: Create system diagrams, define data structures, design user interfaces.

Types: High-level design (architecture), low-level design (modules).

Deliverables: Design document, architecture diagrams, database schema, UI mockups.

Best Practices: Follow design principles (SOLID), consider scalability and maintainability, review designs with peers.

Implementation Phase

Purpose: Convert the design into actual code.
Activities: Write code, conduct code reviews, integrate modules.

Key Aspects: Coding standards, version control, code documentation.

Deliverables: Source code, build scripts, developer documentation.

Best Practices: Use version control (Git), follow coding standards, conduct regular code reviews.

Testing Phase

Purpose: Verify that the software meets requirements and identify defects.
Activities: Write test cases, execute tests, report bugs.

Types: Unit testing, integration testing, system testing, user acceptance testing (UAT).

Deliverables: Test plan, test cases, test reports, bug reports.

Best Practices: Write test cases early, automate testing, track defects, involve end-users in testing.

Deployment and Maintenance

Deployment Phase

Purpose: Release the software to the end-users.
Activities: Prepare environment, install software, migrate data, train users.

Deployment Strategies: Big bang, phased, rolling, blue/green.

Deliverables: Deployment plan, installation scripts, user manuals.

Best Practices: Plan deployment carefully, automate deployment, monitor performance, have a rollback plan.

Maintenance Phase

Purpose: Keep the software running smoothly after deployment.
Activities: Fix bugs, provide support, implement enhancements.

Types: Corrective, adaptive, perfective, preventive.

Deliverables: Bug fixes, updates, new features, maintenance reports.

Best Practices: Track maintenance requests, prioritize fixes, document changes, plan for end-of-life.

Choosing the Right SDLC Model

Factors to Consider

Requirements Clarity: How well-defined are the requirements?
Project Complexity: How complex is the project?
Risk Level: What are the potential risks?
Customer Involvement: How much customer involvement is needed?
Team Expertise: What is the team’s experience with different models?

Model Selection Guide

Waterfall: Use for simple, well-defined projects with stable requirements.
Agile: Use for complex projects with evolving requirements and a need for flexibility.
Spiral: Use for high-risk projects where risk analysis is critical.
Iterative: Use when some requirements are known at the project beginning but evolve as development proceeds.