SOFTWARE DEVELOPMENT LIFE CYCLE

Mansi Thakur

Contents

1. What is SDLC?

The **Software Development Life Cycle (SDLC)** is a step-by-step process that helps teams create software efficiently and effectively. It ensures that the software meets the user's needs, is delivered on time, and stays within the planned budget. Let's explore each phase of the SDLC:

A) Planning Phase

The team decides what they want to achieve and how to do it. This phase is all about preparing for the project.

Activities:

- **Understanding the problem:** What issue are we solving with this software?
- **Setting goals:** What do we want the software to do?
- **Feasibility study:** Can we create this software with our time, skills, and budget?
- **Planning:** Create a timeline, assign tasks, and decide the budget.
- **Talking to stakeholders:** Find out what the users and customers expect.

Outcome:

- A clear plan of action, a budget, and a list of goals and requirements.

B) Requirement Analysis Phase

The team works closely with users and stakeholders to figure out exactly what the software should do.

Activities:

- **Gathering requirements:** Talk to users, customers, and stakeholders to understand their needs.
- **Organizing requirements:** Break down needs into "must-have" (functional requirements) and "good to have" (non-functional requirements).
  Example: A "must-have" could be login functionality; a "good-to-have" could be a dark mode option.
- **Validating requirements:** Double-check if all the requirements make sense and are achievable.
- **Documenting:** Write down everything in a clear document called the Software Requirements Specification (SRS).

Outcome:

- A detailed document that explains what the software will do and how it will behave.

C) System Design Phase

The team creates a blueprint of the software. It shows how the software will look and work.

Activities:

- **High-level design (HLD):** Decide the overall structure of the software (architecture).
  Example: Decide how different parts of the software will communicate, like databases, servers, and users.
- **Low-level design (LLD):** Focus on smaller details, like how a specific feature (e.g., search bar) will work.
- **UI/UX Design:** Create layouts or wireframes to show how the software will look and feel.
- **Selecting tools:** Choose the technologies, programming languages, and tools (e.g., React, Java, MySQL).

Outcome:

Diagrams, designs, and a detailed plan for developers to follow.

D) Implementation (Coding) Phase

Developers start building the software based on the design. This is where coding takes place.

Activities:

- **Write the code:** Developers write code for different parts of the software (frontend, backend, and database).
- **Integrate the code:** Combine all the parts into one complete system.
- **Follow coding standards:** Ensure the code is clean, readable, and efficient.
- **Use version control:** Tools like Git are used to manage code changes and avoid conflicts when working as a team.

Outcome:

A working version of the software that's ready for testing.

E) Testing Phase

The software is tested to make sure it works properly and doesn't have bugs.

Activities:

- **Unit testing:** Test individual features or small pieces of code.
- **Integration testing:** Check if different parts of the software work well together.
- **System testing:** Test the entire software to ensure it meets the requirements.
- **User Acceptance Testing (UAT):** Let users test the software to see if it's what they expected.
- **Automation testing:** Use tools to automate repetitive tests (e.g., Selenium).
- **Bug fixing:** Fix any issues found during testing.

Outcome:

- A tested and bug-free software version.

F) Deployment Phase

The software is delivered to users. It's made available in a real-world environment.

Activities:

- **Prepare the environment:** Set up servers, databases, and other infrastructure for deployment.
- **Deploy the software:** Use tools like Jenkins or Docker to release the software to production.
- **Monitor:** Check if the software is running smoothly after deployment.
- **Provide rollback plans:** Be prepared to revert to the old version if something goes wrong.

Outcome:

The software is live and ready to use.

G) Maintenance Phase

The software is updated, improved, and maintained after it's deployed.

Activities:

- **Bug fixes:** Fix issues reported by users.
- **Updates:** Add new features or improve existing ones based on user feedback.
- **Performance monitoring:** Regularly check the software's performance to ensure it's running well.
- **Adaptation:** Update the software to work with new operating systems, browsers, or hardware.

Outcome:

A stable and updated software that continues to meet user's needs.

2. Why is SDLC Important?

- It helps teams work in an organized way.
- It reduces mistakes and saves time and money.
- It ensures the software meets user expectations.

3. SDLC Models

SDLC models define how software development activities are planned, structured, and executed. Choosing the right model depends on the project's size, complexity, and requirements. Below is a detailed explanation of popular SDLC models, their characteristics, differences, and best use cases.

3.1 Waterfall Model

The Waterfall model is a linear and sequential approach where each phase must be completed before moving to the next. It flows like a waterfall:
Requirements → Design → Implementation → Testing → Deployment → Maintenance.

Key Features:

- Each phase has a defined start and end.
- Phases do not overlap.
- Changes in the middle of development are difficult and costly.

Advantages:

- Easy to understand and manage.
- Works well for smaller projects with clear requirements.
- Clear documentation at each stage.

Disadvantages:

- Not flexible; changes are hard to implement.
- Late discovery of issues since testing happens after development.
- Unsuitable for complex or long-term projects.

Best Used For:

- Small, simple projects with well-defined and unchanging requirements.
- Projects where clarity and predictability are more important than flexibility.

3.2 Agile Model

Agile is an iterative and incremental approach that focuses on flexibility, customer collaboration, and rapid delivery of small, usable features. Work is divided into short cycles called **sprints** (usually 2–4 weeks).

Key Features:

- Continuous customer involvement.
- Deliverables are prioritized, and feedback is incorporated frequently.
- Teams work in short, focused sprints.

Advantages:

- Flexible and adaptable to changing requirements.
- Continuous testing ensures high-quality outputs.
- Encourages teamwork and collaboration.
- Faster delivery of usable features.

Disadvantages:

- Requires experienced team members and strong communication.
- Hard to predict cost and timeline.
- Less focus on documentation.

Best Used For:

- Projects where requirements are unclear or likely to change.

- Startups, mobile app development, and projects with evolving technologies.

3.3 Iterative Model

The Iterative model builds software in small steps (iterations). Each iteration improves on the previous one, adding new features and refining the system.

Key Features:

- Feedback is gathered after each iteration.
- The system evolves over time.
- Focuses on reusability and incremental improvements.

Advantages:

- Identifies risks early by testing in increments.
- Flexible to changes within iterations.
- Delivers a working product early in the cycle.

Disadvantages:

- Requires good planning and design from the start.
- Can be resource-intensive due to multiple iterations.
- May not suit projects with tightly defined requirements.

Best Used For:

- Medium to large projects where requirements are expected to evolve.
- Complex projects requiring risk management, like large enterprise systems.

3.4 Spiral Model

The Spiral model combines iterative development with risk analysis. It's represented as a spiral with four quadrants:

1. **Determine Objectives:** Gather requirements and identify risks.
2. **Risk Analysis:** Assess and mitigate risks.
3. **Engineering:** Develop and test the software incrementally.
4. **Evaluation:** Review and plan for the next iteration.

Key Features:

- Focus on risk management at every stage.
- Cycles through multiple iterations.
- Flexibility to incorporate changes.

Advantages:

- Ideal for high-risk projects.
- Allows for customer feedback early and often.
- Combines the best of iterative and Waterfall models.

Disadvantages:

- Expensive and complex to manage.
- Requires highly skilled teams for risk analysis.
- May extend project timelines.

Best Used For:

- Large, high-budget projects with significant risks, like aerospace and defense systems.
- Projects with unclear or changing requirements.

3.5 V-Model (Validation and Verification)

The V-Model is an extension of the Waterfall model where testing is integrated into every phase. Development moves down one side of the "V" (design) and back up the other (testing).

Key Features:

- Testing activities correspond to development stages.
- Emphasis on validation (are we building the right system?) and verification (are we building the system right?).

Advantages:

- Early detection of defects.
- Clear structure and responsibilities.
- Works well for critical systems requiring high-quality standards.

Disadvantages:

- Inflexible and costly to adapt to changes.
- Requires thorough documentation.
- Testing phases are dependent on completion of development.

Best Used For:

- Projects requiring high reliability and precision, like medical software or embedded systems.

3.6 DevOps Model

DevOps combines development (Dev) and operations (Ops) to ensure faster delivery and smoother maintenance. It emphasizes collaboration, automation, and continuous integration/delivery (CI/CD).

Key Features:

- Continuous integration, testing, delivery, and monitoring.
- Heavy use of automation tools like Jenkins, Docker, and Kubernetes.
- Strong collaboration between development and operations teams.

Advantages:

- Faster delivery of features and fixes.
- Reduced chances of deployment failures.
- Encourages continuous improvement.

Disadvantages:

- Requires significant cultural and organizational changes.
- Heavy reliance on tools and automation.
- Steep learning curve for teams new to DevOps.

Best Used For:

- Projects requiring frequent updates or deployments, such as SaaS applications.
- Large-scale systems with ongoing development and maintenance needs.

4. Comparison of SDLC Models

Aspect	Waterfall	Agile	Iterative	Spiral	V-Model	DevOps
Approach	Linear	Iterative/ Incremental	Iterative	Iterative with Risk	Linear + Testing	Continuous
Flexibility	Low	High	Medium	High	Low	High
Documentation	Extensive	Minimal	Moderate	Moderate	Extensive	Moderate
Risk Management	Low	Medium	Medium	High	Low	Medium
Best For	Small projects	Dynamic projects	Evolving projects	Risky projects	Critical systems	Rapid deployment

5. How to Choose the Right Model

1. **Project Size:**
    - Small projects → Waterfall, V-Model.
    - Large projects → Agile, Spiral, Iterative.
2. **Requirements Clarity:**
    - Clear requirements → Waterfall, V-Model.
    - Evolving requirements → Agile, Spiral.
3. **Risk Level:**
    - Low risk → Waterfall, Agile.
    - High risk → Spiral.
4. **Time Sensitivity:**
    - Fixed timeline → V-Model, Iterative.
    - Flexible timeline → Agile, DevOps.
5. **Team Expertise:**
    - Less experienced team → Waterfall, V-Model.
    - Experienced, collaborative team → Agile, DevOps.