## Software Development Life Cycle

What is it?

Software Development Life Cycle (SDLC) is a process used in the software industry to design, create, and test high-quality software. It aims to meet customer expectations, finish within estimated time and cost.

* **Framework**: It defines tasks at each step of software development. Software development is a multi-step process. Various methods exist but follow a common sequence of steps termed as the Software Development Life Cycle.
* **Standardization:** ISO/IEC 12207 is an international standard for software life-cycle processes.
* The idea of software development dates back to the early days of computers like ENIAC. Practices have evolved with technological advancements, aiming for cost-effective and efficient software creation.

SDLC Functioning

**Iterative Process:** Software development is an ongoing cycle; initial releases are seldom final.

**Feedback Loop:** Error reports and user feedback drive continuous improvements and new feature requests.

Importance of Planning

Methodologies: Different methods exist, each offering a framework for software development.

Necessity of Plans: Structured plans prevent chaos within development teams and enable tracking of progress.

SDLC serves as a roadmap for developing software, ensuring a structured approach that helps in meeting goals, managing teams effectively, and delivering software that meets user requirements.

Benefits of SDLC

* Flexibility and Adaptability
* Structured Approach
* Risk Management
* Consistency and Standardization
* Continuous Improvement (iterative cycles)

## Seven Phases of SDLC



Requirements Analysis/Planning

Involves project and product management.

Resource allocation, capacity planning, scheduling, cost estimation.

Outputs: project plans, schedules, cost estimates, procurement needs.

Defining/Feasibility

Gathering requirements from stakeholders and SMEs.

Checks: Economic, Legal, Operational, Technical, Schedule feasibility.

Outputs: Requirement documents or backlog of tasks.

Design and Prototyping

Software architects design the system based on requirements.

Use of established patterns and frameworks for design.

Outputs: Design documents, code from prototyping.

Coding/Software Development

Actively develops the software based on designs.

Can be time-boxed (Agile) or sequential (Waterfall).

Aim: Produce functional software, engage stakeholders.

Testing

Integral phase to ensure software quality.

Includes code quality, unit, integration, performance, security testing.

Automation crucial; ensures tests are regularly conducted.

Deployment

Ideally automated phase, releasing software to production.

Higher maturity: instant deployment; lower maturity may involve manual steps.

Aim: Full automation through Application Release Automation tools.

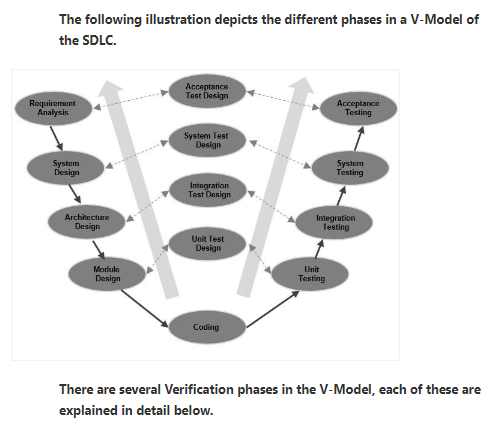
Operations and Maintenance

Continuous phase ensuring proper software operation.

Monitoring, addressing bugs, defect reporting, and responses.

Fixes may not cycle through the entire process; abbreviated processes used.

## SDLC Models: V-Model



V-Model

Sequential SDLC model, resembling a V-shape.

Emphasizes **Verification** and **Validation**.

Each development phase has an associated testing phase.

Highly disciplined and follows a sequential pattern.

V-Model Phases

Business Requirement Analysis

Understand customer needs and expectations.

Detailed communication for clear requirements.

Input for acceptance testing, test design planning.

System Design

Develop the complete system based on requirements.

Detail hardware, communication setup.

Develop system test plans early for test execution.

Architectural Design

Understand architectural specifications.

Propose technical approaches, finalize based on feasibility.

Break down system design into functional modules (High Level Design).

Module Design

Specify detailed internal design for system modules (Low Level Design).

Ensure compatibility within system architecture and external systems.

Design unit tests based on internal module designs to catch errors early.

The V-Model ensures rigorous testing at each stage, aligning with its sequential development approach. Verification phases correspond to development, while Validation phases correspond to testing.

This model's structure emphasizes the importance of early planning and documentation to mitigate errors throughout the software development process.

V-Model Pros and Cons

**Pros**

**Clarity and Structure:** Clear documentation at each stage enhances understanding and reduces ambiguity.

**Early Test Planning:** Tests are planned early, aiding in comprehensive test coverage.

**Traceability:** Easily traceable deliverables from requirements to tests.

**Thoroughness:** Emphasizes testing at every stage, leading to higher product quality.

**Predictability:** Well-defined phases and deliverables offer predictability in project timelines and outcomes.

**Cons**

**Rigidity:** Not adaptable to changes in requirements or scope once a phase is completed.

**Increased Documentation:** Extensive documentation can become time-consuming and costly.

**Limited Client Interaction:** Limited client involvement until later stages might lead to misunderstandings.

**Potential Delay in Deliverables:** Sequential nature can lead to delays in the final product release.

**Risk of Overlooking Requirements:** Early phase commitments might miss crucial requirements discovered later.

## SDLC Models

Spiral Model

**Focuses** on managing risks.

**Adaptability:** Combines elements from various models (like waterfall, incremental).

**Approach:** Merges best aspects of prototyping and waterfall models.

It is an Iterative process, passing through four distinct phases called spirals.

Phases of Spiral Model

1. Identification:

**Start Point:** Gathers initial business requirements.

**Progression:** As the product evolves, it identifies system, subsystem, and unit requirements.

**Customer Engagement:** Continuous communication with customers to understand needs.

**Outcome:** Product deployment in the market.

1. Design

**Starting Point:** Begins with conceptual design.

**Evolution:** Progresses to architectural, logical, and final design in subsequent spirals.

**Involvement:** Crafting designs for system modules and physical product elements.

1. Build

**Objective:** Produces the actual software in each spiral.

**Initial Stage:** Develops a Proof of Concept (POC) for customer feedback.

**Progression:** With clearer requirements, produces working software versions (builds) for customer review.

1. Evaluation/ Risk Analysis

**Risk Assessment:** Identifies, estimates, and monitors technical and management risks.

**Testing and Feedback:** Evaluates the software after testing, gathering customer feedback.

**Outcome:** Iterative improvement based on customer evaluations and risk analysis.

Big Bang Model

**Focus:** Emphasizes coding over planning, suitable for small teams with little or no planning.

**Applicability:** Best for small-scale, academic, or unknown requirement projects.

**Nature:** Limited planning, where requirements are implemented as they arise.

Pros of Big Bang Model:

* **Simplicity:** Requires minimal planning and formal procedures.
* **Ease of Management:** Simple to manage due to its straightforward approach.
* **Resource Efficiency:** Needs very few resources, giving flexibility to developers.
* **Flexibility:** Offers flexibility to developers during the coding phase.
* **Educational Value:** Helpful as a learning tool for newcomers and students.

Cons of Big Bang Model:

* **High Risk:** High risk and uncertainty due to limited planning.
* **Not for Complexity:** Not suitable for complex or object-oriented projects.
* **Inadequate for Long Projects:** Not suitable for long or ongoing projects.
* **Costly Mistakes:** Misunderstood or changing requirements can lead to high expenses or project abandonment.

Waterfall Model

**Structure:** Sequential SDLC model divided into distinct phases.

Suitable for smaller projects with well-understood requirements. Works efficiently when all requirements are clear from the start. Helpful for projects where tasks can be easily arranged and prioritized.

**Phases**: Proceeds linearly from planning through implementation to maintenance.

**Input-Output:** Each phase's output is input for the subsequent phase.

**Rigidity:** Follows a predetermined path, each phase strictly follows the previous one.

**Documentation:** Emphasizes documentation throughout the process.

**Limited Adaptability:** Difficulty in accommodating changing requirements.

A diagram of software components

Description automatically generated

Advantages of Waterfall Model

* **Simplicity:** Easy to understand and use due to its straightforward structure.
* **Manageability:** Rigidity allows for better control and management of each phase.
* **Clarity and Milestones:** Clearly defined stages and milestones aid in planning and tracking progress.
* **Documentation: Process** and results are well-documented, aiding in future references.

Limitations of Waterfall Model:

* **Limited Flexibility:** Inability to adapt to changing requirements during development.
* **Potential for Errors:** Changes after the initial phase may lead to costly errors.
* **Not for Complex Projects:** Less suitable for complex or large-scale projects.
* **No Room for Iteration:** Doesn’t allow revisiting previous phases easily for changes.

Agile Model

**Iterative Development:** Emphasizes continual interaction between development and testing.

**Incremental Builds:** Splits the project into small, manageable builds delivered in short iterations.

**Flexibility:** Adaptable to changing requirements, contrasts with the more rigid Waterfall model.

**Manifesto for Agile:** Drafted in 2001, contrasts Agile with Waterfall's inflexibility.

**Variants**: Various Agile methods exist, including Scrum, Kanban, and others.

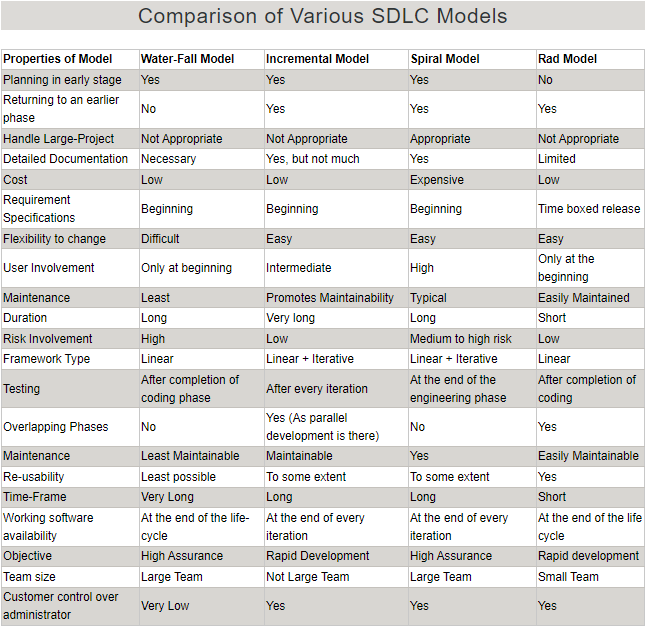
**Principles**: Focuses on collaboration, responsiveness, and delivering working software.

Advantages of Agile Model

* **Realistic Approach:** Reflects real-time software development scenarios.
* **Team Collaboration:** Encourages teamwork and skill diversification.
* **Rapid Development:** Allows for rapid functionality development and demonstration.
* **Minimal Resource Requirements:** Requires fewer resources compared to other models.
* **Adaptability:** Suited for both fixed and evolving requirements.
* **Early Deliveries:** Provides early working solutions for evaluation and feedback.
* **Flexibility and Minimal Documentation:** Offers flexibility to developers, with less reliance on documentation.

Limitations of Agile Model

* **Complex Dependency Handling:** Struggles with complex dependencies in projects.
* **Risk of Sustainability:** May pose challenges in maintaining and extending the software.
* **Need for Agile Practices:** Requires skilled agile leadership and project management.
* **Customer Dependency:** Heavy reliance on customer interaction, which can lead to misdirection.
* **Limited Documentation:** Less documentation might challenge new team members in understanding the technology.



## Summary of SDLC

SDLC Overview

* **Purpose:** Systematic process ensuring software quality and correctness.
* **Full Form:** Software Development Lifecycle (SDLC).
* **Framework:** Provides a structure for standard activities and deliverables.

Seven Stages of SDLC

1. **Requirement Collection and Analysis:**
   * **Conducted by:** Senior team members.
   * **Objective:** Gather and analyze project requirements comprehensively.
2. **Feasibility Study**
   * **Scope:** Encompasses all design and development aspects throughout the project life cycle.
   * **Purpose:** Assess project feasibility across various dimensions.
3. **Design:**
   * **Activities:** Preparation of system and software design documents aligned with requirement specifications.
4. **Coding:**
   * **Tasks:** Developers write code using selected programming languages to build the system.
5. **Testing:**
   * **Objective:** Verify that the application functions as per customer requirements.
   * **Validation:** Ensures software meets expectations and works correctly.
6. **Installation/Deployment:**
   * **Initiation:** Begins after completion of testing phase without residual bugs or errors.
   * **Objective:** Install the software in the intended environment for user access.
7. **Maintenance:**
   * **Activities:** Involves bug fixing, upgrades, and ongoing engagement actions.
   * **Purpose:** Sustain software functionality and address any issues post-deployment.

Popular SDLC Models

* **Waterfall, Incremental, Agile, V model, Spiral, Big Bang:** Represent various SDLC approaches.
* **Diverse Approaches:** Each model offers a different structure and methodology for software development.

SDLC Plan

* **Detailed Plan:** Outlines the process for planning, building, and maintaining specific software.
* **Guidance:** Acts as a guide through the various stages of software development.

**Conclusion:** The SDLC encompasses seven stages, starting from requirement gathering to software maintenance, ensuring a systematic and quality-focused approach to software development. Various models offer different methodologies to suit diverse project needs, all guided by a detailed plan to streamline the development process.

## Prototyping

Software Prototyping

**Definition:** A working model with limited functionality, aiding in user evaluation before full implementation.

Effective for systems with high user interaction like online platforms. Less beneficial for projects emphasizing internal functionality over user interface.

**Purpose:** To understand user-specific requirements, propose solutions, and gather feedback for improvements.

**Process**: Involves basic requirement **identification**, prototype **development**, **review**, **enhancements**, and **iterations**.

Types of Software Prototyping

**Throwaway/Rapid Prototyping:** Minimal effort prototype discarded after understanding actual requirements.

**Evolutionary Prototyping:** Builds functional prototypes forming the basis for the final system.

**Incremental Prototyping:** Develops multiple prototypes for various subsystems, integrating them later into a complete system.

**Extreme Prototyping:** Sequential phases developing a fully functional UI with minimal focus on backend services.

Steps in Designing a Software Prototype

1. **Basic Requirement Identification:** Focuses on understanding initial product requirements, especially concerning user interface elements.
2. **Developing the Initial Prototype:** Showcases basic requirements and interfaces, potentially with workarounds to emulate functionality.
3. **Review of the Prototype:** Presentation to stakeholders and customers for structured feedback collection.
4. **Revise and Enhance the Prototype:** Iterative process based on feedback, incorporating changes until meeting customer expectations.

Best Practices of Prototyping:

* Use when requirements are unclear.
* Controlled and planned prototyping.
* Regular meetings to avoid delays.
* Aware of prototyping challenges.
* Early approval before proceeding.
* Flexibility to change earlier decisions.
* Adequate feature implementation.

Advantages of Prototyping Model:

* Active user involvement reduces error detection in early stages.
* Helps identify missing functionality, reducing the risk of failure.
* Encourages effective team communication.
* Early customer satisfaction with tangible product experiences.
* Quicker feedback leads to better development solutions.
* Facilitates innovation and flexible design.

Disadvantages of Prototyping Model:

* Slow and time-consuming.
* Costly development of prototypes that are ultimately discarded.
* May encourage excessive change requests.
* Challenges in accommodating various requirements.
* Poor documentation due to changing customer needs.
* Potential sub-standard development due to rapid prototype building.