© SEP 2024 | IRE Journals | Volume 8 Issue 3 | ISSN: 2456-8880

A Comparative Analysis of Software Life Cycle Models: Phases, Activities, and Order of Execution

VRUSHALI WADKAR[1], SNEHAL GHORPADE[2]
[1, 2]*Department of Computer Science, Karmveer Bhaurao Patil College*

Abstract- Software development life cycle (SDLC) models offer a systematic approach to software development, guiding from conception to deployment and maintenance [4]. This paper compares five prominent SDLC models: Waterfall, V-Model, Incremental, Spiral, and Agile, analyzing their phases, activities, and execution order, to provide insights into selecting the most suitable model for a project. The analysis focuses on the phases, activities, and the order of execution inherent in each model, highlighting their respective strengths and weaknesses. The comparison aims to offer guidance on selecting the most suitable model based on project requirements, risks, and flexibility needs.

Indexed Terms- Software Development Life Cycle (SDLC), Waterfall, V-Model, Incremental Model, Spiral Model, Agile Model, Phases, Activities, Order of Execution.

I. INTRODUCTION

Software development is a complex process that requires careful planning and execution. SDLC models have been developed to structure this process, each offering unique approaches to managing the various stages of software development. The choice of an SDLC model can significantly impact the success of a project, making it crucial to understand the differences between them [3], [4]. The analysis focuses on their phases, activities, and execution order, highlighting the importance of understanding the differences between them.

1. Waterfall Model

1.1 Overview

The Waterfall model is one of the earliest and most traditional software development life cycle (SDLC) methodologies. It follows a linear and sequential approach, where each phase of the development process is completed before the next phase begins [1]. This model is characterized by its structured, stage-by-stage progression through the phases of requirements gathering, system design, implementation, testing, deployment, and maintenance.

1.2 Phases and Activities

Requirement Analysis: Gathering detailed software requirements. System Design: Transforming requirements into a system architecture. Implementation: Coding the software based on the design. Integration and Testing: Combining and testing all components. Deployment: Delivering the final product to the client. Maintenance: Providing ongoing support and updates [1].

1.3 Order of Execution

- Strictly sequential with clear start/end.
- Avoids backtracking for cost and difficulty [1].

1.4 Strengths and Weaknesses

Strengths:

Simplicity and Ease of Use: The Waterfall model is straightforward, with clearly defined stages and milestones. Its linear approach is easy to understand and implement, especially for small projects with well-defined requirements [1].

Structured Documentation: This model emphasizes documentation at each stage, which provides a clear and detailed record of the project's progress and decisions. This can be particularly useful for maintenance and future project phases [1]. Ease of Management: The sequential nature of the Waterfall model makes it easier to manage since the project progresses through well-defined phases with specific deliverables and milestones [1].

Weaknesses:
Rigidity: The Waterfall model is inflexible when it comes to handling changes. Once a phase is completed, it is difficult and costly to go back and make changes [1].
Late Testing: Testing only occurs after the development phase, meaning that issues or defects are often detected late in the process, which can lead to expensive and time-consuming fixes [1].
Poor Suitability for Complex or Evolving Projects: The model assumes that all requirements can be identified upfront, which is often not the case in complex or evolving projects. This can result in significant rework or project failure if requirements change [1].

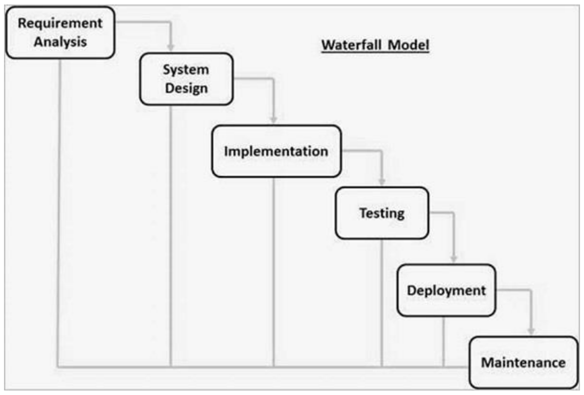


Image 1: Waterfall Model

II. V-MODEL

2.1 Overview
The V-Model, also known as the Verification and Validation Model, is a software development life cycle (SDLC) framework that extends the traditional Waterfall model by emphasizing a parallel relationship between development and testing activities [6]. Structured as a "V" shape, the model highlights the correlation between each development phase and its corresponding testing phase, thereby ensuring that verification and validation are integral parts of the process.

2.2 Phases and Activities Requirement Analysis: Understanding and documenting requirements. System Design: High-level design of the system architecture. High-Level Design (HLD): Designing system components and interfaces. Low-Level Design (LLD): Detailed component design. Implementation: Coding based on the detailed design. Unit Testing: Testing individual components. Integration Testing: Testing the integration of components. System Testing: Validating the entire system. Acceptance Testing: Ensuring the system meets user requirements [6].

2.3 Order of Execution Sequential Parallel focus on testing activities. Forms V-shape [6].

2.4 Strengths and Weaknesses Strengths:
Emphasis on Testing: The V-Model incorporates testing at each stage of development, ensuring that verification and validation are conducted early in the process. This can lead to higher-quality outputs and fewer defects [6]. Clear and Structured Approach: Like the Waterfall model, the V-Model has a structured approach with well-defined phases, making it easy to manage and implement [6].
Strong Documentation: The model provides detailed documentation throughout the project, which is beneficial for future reference and project continuity [6].

Weaknesses:
Inflexibility: Similar to the Waterfall model, the V-Model is rigid, making it difficult to accommodate changes once a phase is completed [6].
High Cost of Changes: Any changes that are needed require revisiting and revising earlier phases, which can be costly and time-consuming [6].
Not Suitable for Uncertain or Evolving Requirements: The V-Model works best when requirements are well understood from the outset. It is less effective for projects where requirements are expected to evolve over time [6].

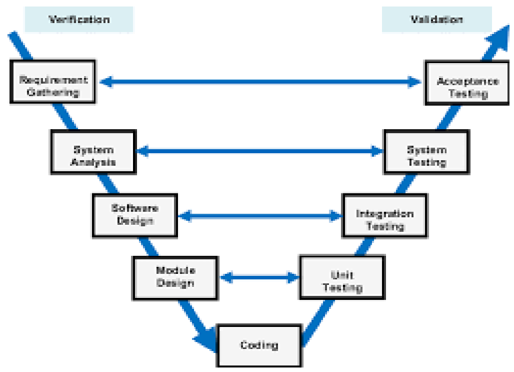


Image 2: V-Model

III. INCREMENTAL MODEL

3.1 Overview
The Incremental Model is a software development approach that divides the project into smaller, manageable segments or increments, each of which is developed and delivered sequentially [3]. Unlike traditional linear models, the Incremental Model allows for the progressive development of a system by building upon previous iterations. Each increment adds functional capabilities, providing partial system functionality early in the development process and enabling continuous user feedback [3].

3.2 Phases and Activities Requirement Analysis: Prioritizing and breaking down requirements.
System Design: Developing an overall architecture.
Implementation and Testing: Developing, integrating, and testing in increments.

Incremental Integration: Gradually adding new functionality. Deployment: Deploying the system once usable increments are complete.
Maintenance: Providing ongoing support as new increments are added [3].

3.3 Order of Execution Cyclic and iterative Each increment passes through design, development, and testing phases [3].

3.4 Strengths and Weaknesses Strengths: Flexibility: The Incremental Model allows for partial implementation of the system and adds features or modules in increments, making it more flexible than the Waterfall or V-Model [3].

Early Functionality: Since the system is developed in increments, the user gets to see some functionality early in the development process, allowing for early feedback and adjustments [3].

Risk Management: By breaking the project into smaller, manageable increments, the model reduces risks and allows for easier management of changes [3].

Weaknesses:
Integration Challenges: Combining the increments to form a complete system can be challenging and may lead to integration issues [3].

Complexity: Managing multiple increments can add complexity to the project, requiring careful planning and coordination [3].

May Lead to Scope Creep: Since the project is developed in increments, there is a risk of scope creep as users may request additional features that were not originally planned [3].

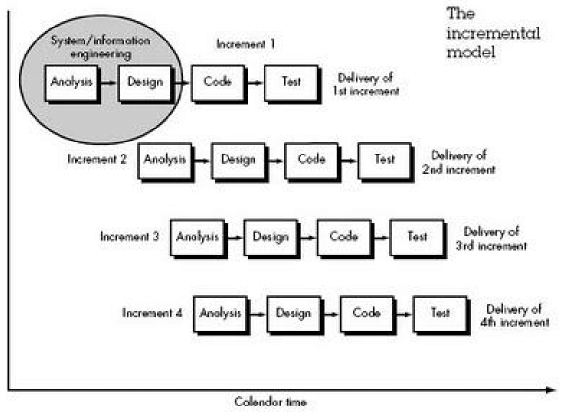

The incremental model
System/information engineering
Increment 1
Analysis
Design
Code
Test
Delivery of 1st increment
Increment 2
Delivery of 2nd increment
Increment 3
Delivery of 3rd increment
Increment 4
Delivery of 4th increment
Calendar time

Image 3: Incremental Model

IV. SPIRAL MODEL

4.1 Overview
The Spiral Model is structured as a series of iterative cycles, each of which involves four key phases: planning, risk analysis, engineering, and evaluation [2]. At the beginning of each cycle, project goals and constraints are defined during the planning phase. This is followed by a rigorous risk analysis phase, where potential risks are identified, assessed, and mitigation strategies are developed [2]. The engineering phase then focuses on the actual development and testing of the software, while the evaluation phase involves customer feedback and the refinement of the project [2].

4.2 Phases and Activities Planning: Defining objectives, alternatives, and constraints.
Risk Analysis: Identifying and mitigating risks. Engineering: Developing and testing the software. Evaluation: Reviewing the progress and planning the next iteration [2].

4.3 Order of Execution Iterative with multiple cycles. Includes planning, risk analysis, engineering, evaluation [2].

4.3 Strengths and Weaknesses Strengths: Risk Management: The Spiral Model is highly effective in managing risks, as it emphasizes risk analysis and mitigation at each iteration of the project [2].

Flexibility: The model is adaptable to changes and can accommodate evolving requirements, making it suitable for complex and large-scale projects [2]. Continuous Refinement: Through iterative cycles, the model allows for continuous refinement of the product, leading to higher quality and better alignment with user needs [2].

Weaknesses:
Complexity: The Spiral Model is more complex to manage and implement compared to other models, requiring expertise in risk assessment and management [2].

High Cost: The iterative nature and extensive risk management activities can make the model expensive, especially for small or less critical projects [2].

Difficult to Manage: Due to its complexity and flexibility, managing the Spiral Model can be challenging, requiring highly skilled project managers [2].

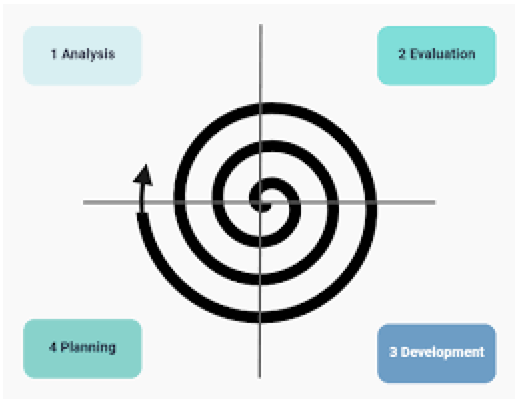


Image 4: Spiral Model

V. AGILE MODEL

5.1 Overview
The Agile model is a dynamic and iterative approach that allows for continuous refinement and adaptation throughout the development process [5]. It is particularly suited for projects with evolving requirements, where customer needs and market conditions are likely to change [5].

5.2 Phases and Activities Requirement Gathering: Collecting and prioritizing user stories. Iteration Planning: Planning short development cycles (sprints).
Design and Development: Designing, coding, and testing in each sprint.
Testing: Continuous testing throughout the development cycle.
Review: Reviewing progress with stakeholders after each sprint.
Deployment: Potentially deploying after each sprint.
Maintenance: Ongoing adaptation and support based on feedback [5].

5.3 Order of Execution
- Iterative with short cycles.
- Allows continuous adaptation and frequent releases [5].

5.4 Strengths and Weaknesses Strengths:
High Flexibility and Adaptability: Agile is highly adaptable to changes, allowing teams to respond quickly to evolving requirements and market conditions [5].

Customer Collaboration: Agile emphasizes close collaboration with the customer, ensuring that the final product closely meets user needs and expectations [5].

Early and Continuous Delivery: Agile promotes the early and continuous delivery of functional software, allowing users to benefit from the product sooner and providing frequent opportunities for feedback [5].

Weaknesses:
Less Predictable: The iterative nature of Agile can make it difficult to predict timelines, costs, and

project scope, especially in the early stages [5]. Requires Strong Team Collaboration: Agile relies heavily on effective communication and collaboration within the team, which can be challenging if the team is not co-located or lacks experience with Agile practices [5]. Scope Creep Risk: The flexibility of Agile can lead to scope creep if changes are not carefully managed, potentially resulting in delays or budget overruns [5].

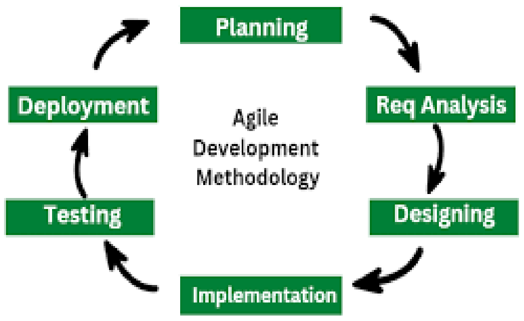
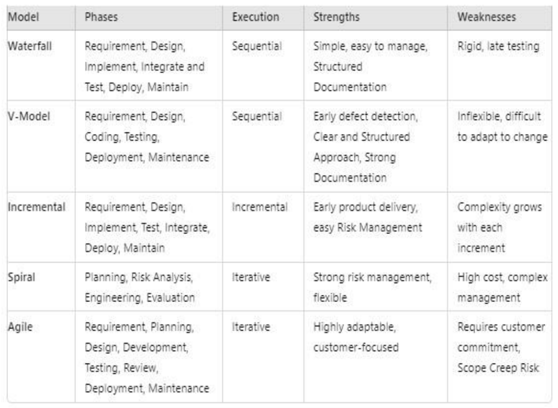


Image 5: Agile Model

Comparative Analysis:



Model	Phases	Execution	Strengths	Weaknesses
Waterfall	Requirement, Design, Implement, Integrate and Test, Deploy, Maintain	Sequential	Simple, easy to manage, Structured Documentation	Rigid, late testing
V-Model	Requirement, Design, Coding, Testing, Deployment, Maintenance	Sequential	Early defect detection, Clear and Structured Approach, Strong Documentation	Inflexible, difficult to adapt to change
Incremental	Requirement, Design, Implement, Test, Integrate, Deploy, Maintain	Incremental	Early product delivery, easy Risk Management	Complexity grows with each increment
Spiral	Planning, Risk Analysis, Engineering, Evaluation	Iterative	Strong risk management, flexible	High cost, complex management
Agile	Requirement, Planning, Design, Development, Testing, Review, Deployment, Maintenance	Iterative	Highly adaptable, customer-focused	Requires customer commitment, Scope Creep Risk

Image 6: Comparative analysis of all SDLC Models

CONCLUSION

The selection of an SDLC model is crucial for a successful software project. Each model possesses unique characteristics that render it suitable for specific types of projects [2], [3], [5]. The Waterfall and V-Model are ideal for projects with well-defined requirements and minimal changes [1], [6]. In contrast, the Incremental, Spiral, and Agile models offer greater flexibility, making them more appropriate for complex and evolving projects [2], [3], [5]. Understanding the phases, activities, and sequential execution inherent in these models helps in making informed decisions that align with the project's goals and constraints.

REFERENCES

[1] W. Royce, “Managing the Development of Large Software Systems,” in Proceedings of IEEE WESCON, 1970, pp. 1–9.

[2] B. Boehm, “A Spiral Model of Software Development and Enhancement,” ACM SIGSOFT Software Engineering Notes, vol. 11, no. 4, pp. 14-24, 1986.

[3] J. W. Satzinger, R. B. Jackson, and S. D. Burd, Systems Analysis and Design in a Changing World, 7th ed. Boston, MA: Cengage Learning, 2015.

[4] I. Sommerville, Software Engineering, 10th ed. Boston, MA: Pearson, 2015.

[5] K. Beck et al., Manifesto for Agile Software Development, 2001. [Online]. Available: https://agilemanifesto.org/.

[6] M. R. Lyu, “Software Reliability Engineering: A Roadmap,” in Future of Software Engineering, IEEE Computer Society, 2007, pp. 153–170.