Crimson Publishers
Wings to the Research

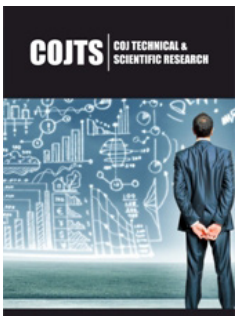
Mini Review

A Short Review on Software Development Life Cycles

Samia Sharmin Diba and Hitesh Mohapatra*

School of Computer Engineering, KIIT University, India

ISSN: 2643-7066



***Corresponding author:** Hitesh Mohapatra, School of Computer Engineering, KIIT (Deemed to be) University, Bhubaneswar -751024, Odisha, India

Submission: July 25, 2023

Published: October 02, 2023

Volume 4 - Issue 4

How to cite this article: Samia Sharmin Diba and Hitesh Mohapatra. A Short Review on Software Development Life Cycles. COJ Tech Sci Res. 4(4). COJTS. 000594. 2023.
DOI: 10.31031/COJTS.2023.04.000594

Introduction to Software Development Life Cycle

The Software Development Life Cycle (SDLC) can be defined as a comprehensive and visual representation of the entire journey of a software product, starting from its inception and continuing until its retirement or end of usefulness. It encompasses all the necessary activities that need to be undertaken throughout the software's life cycle, depicting the order in which these activities are performed. The SDLC provides a descriptive and diagrammatic representation of the various stages involved in the development, deployment and maintenance of a software product. It serves as a roadmap, outlining the sequential execution of tasks, processes and methodologies employed to ensure the successful creation, evolution, and eventual retirement of the software. Essentially, the SDLC captures the entire spectrum of activities performed on a software product, guiding its development and management from beginning to end.

The SDLC typically consists of the following stages:

Requirements gathering

In this phase, the project requirements are collected by communicating with stakeholders, users, and domain experts. The goal is to understand the needs and expectations of the software.

Design

The design phase involves creating a blueprint for the software solution. This includes designing the architecture, database structure, user interface, and other system components. The design should address the functional and non-functional requirements of the software.

Development

In this phase, the actual coding of the software takes place. Developers write code based on the design specifications and programming languages. This stage may involve multiple iterations and testing to ensure the software meets the desired functionality.

Testing

The developed software undergoes various testing processes to identify and fix bugs, errors, and other issues. This includes unit testing, integration testing, system testing, and user acceptance testing. The goal is to ensure that the software functions correctly and meets the specified requirements.

Deployment/Implement

Once the software has been thoroughly tested and approved, it is deployed to the production environment. This involves installing the software on the target systems and making it available to end users. Configuration and data migration may also be performed during this phase.

Maintenance

After deployment, the software requires ongoing maintenance and support. This includes monitoring its performance, addressing user feedback, fixing bugs, and releasing updates or patches as needed. Maintenance may also involve enhancements or new feature development based on evolving user needs (Figure 1).

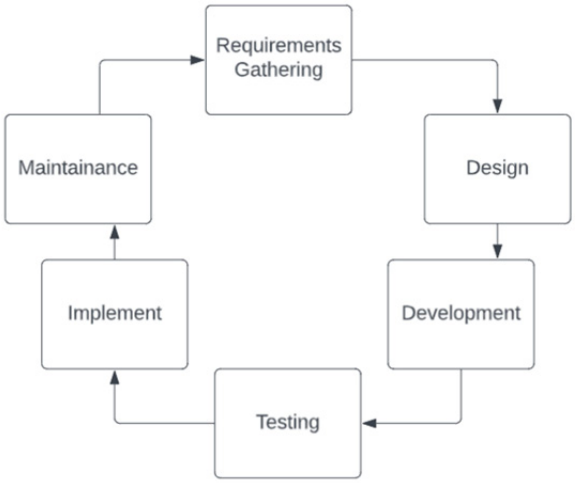


Figure 1: SDLC Model.

Review on Existing on SDLC Models

Explaining 10 existing different SDLC models with their strength and Weakness

Waterfall model

The Waterfall Model follows a sequential approach, with each phase being completed before moving on to the next. It starts with requirements gathering, followed by design, development, testing, deployment, and maintenance. The waterfall model provides a structured and systematic approach, making it easy to understand and manage. It is suitable for projects with stable and well-defined requirements. Its rigid sequential nature makes it difficult to accommodate changes or incorporate feedback during development. It may lead to a lengthy development cycle and delays in addressing issues [1].

Agile model

Agile methodologies, such as Scrum and Kanban, prioritize flexibility and adaptability. It involves iterative development, with small increments called sprints, where requirements are continuously refined, developed, and tested. Agile methodologies prioritize flexibility, collaboration, and iterative development. They enable quick adaptation to changing requirements and encourage customer involvement throughout the process. Agile requires active communication and coordination within the team and with stakeholders [2]. It may be challenging to scale for large and complex projects, and documentation can be less comprehensive.

Spiral model

The Spiral Model combines elements of both waterfall and iterative development. It includes repeated cycles of prototyping, evaluation, and refinement, with each cycle addressing identified risks. The spiral model allows for early risk identification and mitigation, making it suitable for projects with high uncertainties. It incorporates prototyping and frequent evaluations. It can be time-consuming and costly due to repeated cycles of prototyping and evaluation. Its complexity requires experienced project management, and it may not be ideal for small projects [3].

D.V model

The V-Model is an extension of the waterfall model that emphasizes testing at each phase. It associates a testing phase with each corresponding development phase, ensuring comprehensive quality assurance. The V-model emphasizes testing at each phase, ensuring comprehensive quality assurance. It promotes early defect identification and reduces rework. It can be rigid and inflexible, making it difficult to accommodate changes. It assumes that all requirements are known upfront, which may not be realistic in some projects [4].

Iterative model

The Iterative Model involves developing software in small increments or iterations, each going through the entire SDLC. Each iteration results in a working product increment, enabling faster delivery of value to stakeholders. The iterative model supports flexibility, adaptation, and feedback incorporation. It allows for the delivery of incremental value, reduces project risks, and encourages stakeholder involvement. Managing multiple iterations requires careful planning and prioritization. It can be challenging to strike a balance between delivering iterations and addressing changing requirements [5].

Rapid Application Development (RAD)

RAD focuses on quickly delivering a working software application. It involves prototyping, iterative development, and active stakeholder involvement to accelerate development and ensure customer satisfaction. RAD accelerates development by emphasizing prototyping and iterative delivery. It fosters stakeholder engagement, quick feedback, and faster time-to-market. It may not be suitable for large-scale projects with complex requirements. The focus on speed may compromise thorough documentation and may lead to increased technical debt [6].

Incremental model

The Incremental Model divides the software into modules or functional components. Each increment builds upon the previous one, allowing for early delivery of usable features and frequent feedback incorporation. The incremental model enables early and frequent releases, providing value to users sooner. It supports parallel development of different modules and allows for flexibility in prioritization. It requires careful planning and coordination between modules. It may result in a fragmented system if dependencies between modules are not managed properly [7].

DevOps model

DevOps integrates development (Dev) and Operations (Ops) teams to streamline the software development lifecycle. It

emphasizes continuous integration, delivery, and deployment, with a strong focus on collaboration and automation. DevOps promotes collaboration, automation, and continuous integration and delivery. It streamlines the development and deployment process, allowing for faster and more reliable releases (Table 1). It requires significant cultural and organizational changes to implement successfully. The focus on automation may overlook certain aspects, such as security and regulatory compliance [8].

Table 1: Comparison among existing algorithms.

SDLC Model	Flexiblit y	Adaptablit y to Change	Customer Involvem Ent	Development Speed	Risk Management Nt	Cost
WATERFALL	low	low	low	moderate	moderate	high
AGILE	high	high	high	high	low	moderate
SPIRAL	moderat	moderate	moderate	moderate	high	high
V	low	low	low	moderate	moderate	high
ITERATIVE	moderat	high	moderate	moderate	moderate	moderate
RAD	moderate	low	moderate	high	low	low
INCREMENTAL	high	moderate	high	moderate	low	moderate
DEVOPS	high	high	high	high	moderate	high
LEAN	high	high	moderate	moderate	high	moderate
PROTOTYPE	low	moderate	high	high	low	low

Lean development

Lean Development aims to eliminate waste and optimize efficiency. It focuses on value delivery, customer satisfaction, and continuous improvement by minimizing non-value-added activities and maximizing productivity. Lean development focuses on efficiency, waste reduction, and continuous improvement. It emphasizes value delivery, customer satisfaction, and optimization of processes. Lean development requires disciplined execution and ongoing monitoring to identify and eliminate waste effectively. It may face resistance in organizations that are not accustomed to Lean principles [9].

Prototype model

The Prototype Model involves building a basic working version of the software to gather feedback and refine requirements. It helps validate concepts, demonstrate functionality, and improve understanding between stakeholders and developers. The prototype model allows for early validation of concepts and requirements. It helps gather feedback, refine designs, and improve user satisfaction. Managing expectations can be challenging, as stakeholders may confuse the prototype with the final product. There is a risk of the prototype not aligning with the technical constraints of the final product [10].

Flexibility and adaptability to change

Agile, Incremental, DevOps, and Lean Development models provide high flexibility and adaptability to changing requirements. Customer Involvement: Agile, Incremental, and Prototype models emphasize high customer involvement throughout the development process. Development speed: Agile, Rapid Application Development, DevOps, and prototype models focus on faster development speed. Risk Management: Spiral and Lean Development models have a strong focus on risk management and early identification and mitigation of risks. Documentation: Waterfall and V-Model models prioritize comprehensive documentation, while Rapid Application Development and Lean Development models have a lower emphasis on documentation. Cost: waterfall, Spiral, DevOP, V shape models have high price cost than other models [11].

Suggested Models

Here's a suggestion for an improved SDLC model that combines the strengths of different existing models:

Requirements gathering

Gather and document the software requirements from stakeholders.

Agile development (Iterations)

Implement an Agile approach, using iterations (sprints) to develop and deliver working software increments. Emphasize collaboration, flexibility, and customer involvement [12].

Risk assessment and mitigation

Conduct continuous risk assessment throughout the development process and implement appropriate mitigation strategies to manage project risks effectively.

Incremental development

Build the software incrementally, focusing on developing and delivering specific features or modules in each iteration.

Continuous testing and quality assurance

Implement a continuous testing approach, performing comprehensive testing at each iteration. Ensure continuous quality assurance to identify and address defects early [13].

Deployment and continuous integration

Deploy the software to the production environment or make it available to end-users (Figure 2). Implement continuous integration practices to streamline the integration of new features and updates [14].

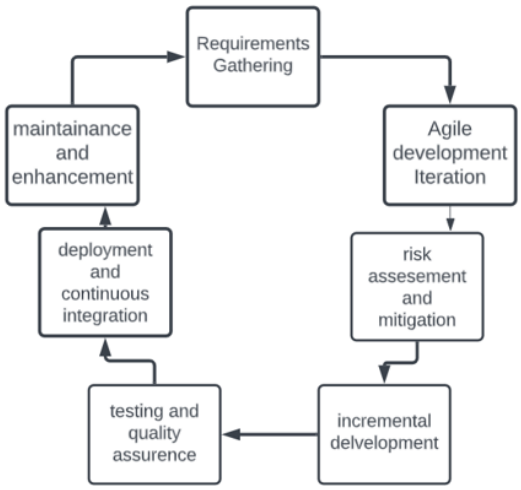


Figure 2: Modified SDLC Model: Hybrid Model

Maintenance and enhancement

Provide ongoing maintenance and support for the software. Incorporate user feedback and make enhancements as needed. The Hybrid Model combines the iterative and flexible nature of Agile with risk management, incremental development, continuous testing, and deployment practices. It aims to balance adaptability, risk mitigation, and continuous delivery, allowing for a more efficient and effective software development process. However, it's important to customize and adapt the model according to the specific needs and constraints of each project [15]. The suggested Hybrid Model, which combines the strengths of different SDLC models, can be particularly helpful in the current era of software development for several reasons:

Flexibility and adaptability

The Hybrid Model incorporates Agile development principles, allowing for flexibility and adaptability to changing requirements. In the current era where technology and business needs evolve rapidly, having a flexible approach enables teams to respond quickly to new market demands and stakeholder feedback.

Continuous customer involvement

The Hybrid Model emphasizes customer involvement throughout the development process. In today's customer-centric market, engaging customers and stakeholders throughout the software development lifecycle helps ensure that the final product meets their needs and expectations.

Risk Management

The Hybrid Model incorporates risk assessment and mitigation practices. In the current era, where projects often face complex challenges and uncertainties, effective risk management is crucial for minimizing potential disruptions and ensuring successful project outcomes.

Continuous testing and quality assurance

The Hybrid Model promotes continuous testing and quality assurance practices. With the increasing complexity of software applications and the need for rapid releases, continuous testing ensures that issues and defects are identified and addressed early in the development cycle, leading to higher quality software products.

Incremental development and continuous deployment

The Hybrid Model integrates incremental development and continuous deployment practices. This allows for faster time-to-market, enabling organizations to release working software increments to end-users more frequently. In the current era of agile business environments, rapid delivery of value to customers is a significant competitive advantage.

Collaboration and continuous improvement

The Hybrid Model encourages collaboration among team members and stakeholders, fostering a culture of continuous improvement. By actively seeking feedback, embracing cross-functional collaboration, and promoting knowledge sharing, the model facilitates ongoing learning and optimization of development processes.

Customization and adaptation

The Hybrid Model recognizes the importance of tailoring the development process to fit specific project needs and constraints. This adaptability is essential in the current era, where diverse projects, technologies, and teams require flexible approaches that can be customized to address unique requirements effectively. In summary, the suggested Hybrid Model aligns with the current era of software development by embracing agility, customer-centricity, risk management, continuous testing, and deployment. It offers a versatile framework that can be customized to suit different projects and helps organizations navigate the challenges and opportunities of today's dynamic and competitive software development landscape. Here are the benefits and limitations of the proposed Hybrid Model:

Benefits of the Hybrid Model

Flexibility and Adaptability

The Hybrid Model combines the flexibility of Agile development with other SDLC practices, allowing teams to adapt to changing requirements and business needs efficiently.

Customer Involvement

The model emphasizes continuous customer involvement, ensuring that the software aligns with customer expectations and delivers maximum value.

Risk mitigation

The Hybrid Model incorporates risk assessment and mitigation practices, enabling early identification and mitigation of potential risks and issues.

Incremental development and continuous deployment

The model supports incremental development and continuous deployment, enabling faster time-to-market and regular delivery of working software increments to end-users.

Continuous testing and quality assurance

The Hybrid Model promotes continuous testing and quality assurance practices, leading to higher software quality and reliability.

Collaboration and continuous improvement

The model encourages collaboration among team members and stakeholders, fostering a culture of continuous improvement and innovation.

Customization and Adaptation

The Hybrid Model can be customized and adapted to fit specific project requirements, team dynamics, and organizational contexts.

Limitations of the Hybrid Model

Complexity

The Hybrid Model may introduce additional complexity due to the integration of multiple practices and methodologies. This complexity can require skilled project management and coordination.

Resource allocation

Adopting a hybrid approach may require allocation of appropriate resources, including skilled personnel and tools, to effectively implement and manage different practices within the model.

Learning curve

Transitioning to a hybrid model may require a learning curve for teams, especially if they are not familiar with all the incorporated practices. Adequate training and support are necessary for successful implementation.

Balancing trade-offs

Balancing trade-offs between agility and risk management can be challenging. It requires careful decision-making to optimize development speed while mitigating risks effectively.

Organizational readiness

The success of the Hybrid Model depends on the readiness and adaptability of the organization to embrace and support the necessary changes in processes, culture, and collaboration.

Potential overhead

Implementing multiple practices within the Hybrid Model may introduce additional overhead, such as increased documentation and coordination efforts. Proper planning and streamlined processes are necessary to minimize overhead.

Context dependency

The suitability of the Hybrid Model may vary depending on the project size, complexity, team size, and other contextual factors. It may require tailoring to fit specific project needs.

Conclusion

SDLC encompasses various models, each with unique strengths and weaknesses. Waterfall Model provides structure but lacks flexibility and adaptability. Agile Methodologies prioritize flexibility, adaptability, and customer involvement. Spiral Model is suitable for uncertain projects and focuses on risk management. V-Model emphasizes comprehensive testing but can be inflexible. Iterative Model offers flexibility, adaptability, and incremental value delivery. Rapid Application Development accelerates development but may not be ideal for large-scale projects. Incremental Model enables early releases and flexible prioritization. DevOps emphasizes collaboration, automation, and continuous integration/deployment. Lean Development focuses on efficiency, waste reduction, and continuous improvement. Prototype Model allows for early validation of requirements through prototyping. Hybrid Model combines strengths, providing flexibility, customer involvement, risk management, incremental development, continuous testing, and deployment. Benefits of Hybrid Model include adaptability, customer-centricity, risk mitigation, faster delivery, and continuous improvement. Limitations of Hybrid Model include complexity, resource allocation, learning curve, trade-off balancing, organizational readiness, potential overhead, and context dependency. Customization is essential for successful implementation of the Hybrid Model. Waterfall Model is suitable for projects with stable requirements. Agile methodologies offer flexibility and adaptability to changing requirements. Spiral Model focuses on risk management and early issue identification. V-Model emphasizes comprehensive testing and quality assurance. The choice of SDLC model should be based on project requirements, team dynamics, and organizational context.

References

1. Jovanovi M (2009) Software testing methods and techniques. IPSI BgD Journals 5: 30-41.
2. Manjit K, Raj K (2011) Comparative study of automated testing tools: Test complete and quicktest pro. International Journal of Computer Application 24: 1-3.
3. Ian Sommerville (2004) Software Engineering.
4. Nabil MAM, Govardhan A (2010) A comparison between five models of software engineering. IJCSI International Journal of Computer Science 7(5): 1694-0814.
5. Roger SP (2023) Software Engineering a Practitioner's Approach.
6. Whitgift D (1991) Methods and tools for software configuration management.
7. Petersen K, Wohlin C, Baca D (2009) The Waterfall Model in Large-Scale Development. P. 32.
8. Shubhmeet Kaur (2015) A review of software development life cycle models. In International Journal of Advanced Research in Computer Science and Software Engineering 5(11).
9. Boehm B (1985) A spiral model of software development and enhancement. In Proceedings of an International Workshop on the Software Process and Software Environments.
10. De Grace, Peter, S, Leslie H (1990) Wicked problems, righteous solutions: A catalogue of modern software engineering. New Jersey.
11. Coad P, Edward Y (1991) Object-Oriented Design.
12. Software Technology Support Centre (1993) Software Management Guide 1: 23.
13. Dyer M (1993) The Cleanroom Approach to Quality Software Development.
14. Blum BI (1992) Software Engineering: A holistic View.
15. Booch G (1994) Software Engineering with Ada. pp. 25