Traditional Software Development Models Vs Agile

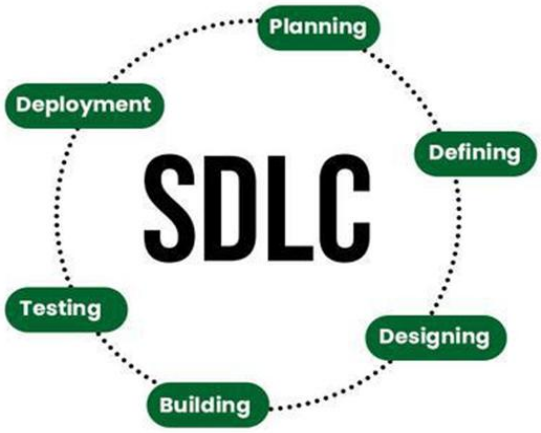
Md Jasim Ansari[1], Dr. Vivek Patil[2]
[1,2]*M. Tech Computer Engineering Dy Patil University, Ambi*

***Abstract*—This Paper will focus on traditional SDLC method like Waterfall model, prototype model and will brief on comparative study on same with Agile.**
Traditional Software Development is the Software development process used to design and develop simple software. It is used when the security and many other factors of the software are not much important. It is used by freshers to develop the software.
Agile Software Development is the software development process used to design complicated software. It is used when the software is quite sensitive and complicated. It is used when security is much more important. It is used by professionals to develop the software.
The primary data collection method was interviews of the industry expertise. The secondary source of data is reference books and Internet articles. This paper will help to understand basics of Agile methodology.

***Index Terms*—Software Development Lifecycle (SDLC), Software Development, Software, Waterfall model, Prototype model, Agile, Agile methods, Agile-methodology.**

I. INTRODUCTION

Software Development Life cycle (SDLC) is the process consisting of a series of planned activities to develop or alter the software products. SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process. The figure 1.1 is a graphical representation of the various stages of a typical SDLC. Each phase of SDLC ensures that the end product is able to meet the customer's expectations and fits within the overall budget.



1.1. Phases of life cycle model

Among the most debated topics in software engineering is the choice between Agile methodology and traditional software development approaches, most commonly represented by the Waterfall model. As technology advances, companies face increasing pressure to deliver high-quality software products within strict deadlines and budgets. Agile methodology emerged as a response to the rigidity of traditional approaches, offering flexibility and iterative development cycles. This study aims to compare Agile and traditional methodologies focusing on efficiency, adaptability, team productivity, and software quality.

II. TRADITIONAL LIFE CYCLE MODEL

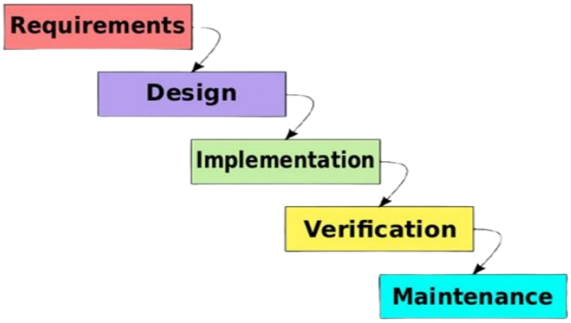
Following are the most important and popular SDLC models followed in the industry:

1. Waterfall Model
2. Spiral Model
3. V-Model
4. Big Bang Model

2.1 Waterfall Model

Waterfall Model is one of the most popular development models used in the software industry.

The original version of this model was first presented by Winston Royce in 1970. Royce described a model structure around phases and argued that the sequential nature of the model is unrealistic and flawed. He stated that iterations are needed after each phase to deal with corrections before proceeding to the subsequent phase.



The Waterfall Model follows a linear, phase-by-phase approach, where each phase must be completed before moving to the next. It does not allow backtracking and permits only minimal changes once a phase is completed. The model is simple, well-structured, and easy to manage, offering high predictability and clearly defined milestones throughout the development process

2.1.1 Features of Waterfall Model

- Development follows a linear, step-by-step process where each phase is completed before moving to the next.
- Detailed documentation is created at every stage to clearly define requirements, design, and progress.
- Detailed documentation is created at every stage to clearly define requirements, design, and progress.
- The project scope, schedule, and deliverables are carefully planned and monitored throughout the lifecycle.

2.1.2 When to use

- Requirements are clear, stable, and fully documented before development begins.
- The project scope is unlikely to change during development.
- Suitable for projects with limited complexity and a clear development path.
- Risks are known, manageable, and can be addressed early in the development cycle.
- Best when the client wants a step-by-step, linear development process.
- Works well when resources are limited and need careful planning and allocation.

2.1.2 Advantages of Waterfall Model

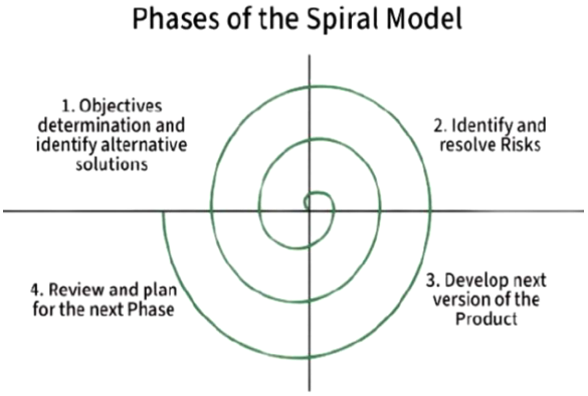
- The model is straightforward and simple, making it easy to learn and follow.
- Each phase is completed one at a time, ensuring an organized development flow.
- Every phase has clear objectives and deliverables, reducing confusion.
- Works well for smaller projects where requirements are clearly understood and unlikely to change.

2.1.3 Disadvantages

- The problems with one phase are never solved completely during that phase and in fact many problems regarding a particular phase arise after the phase is signed off, this result in badly structured system.
- If client want the requirement to be changed, it will not be implemented in the current development process.

2.2 Spiral Model

The Spiral Model for the software development was introduced by Barry Boehm in 1988. The model addresses the weakness of the waterfall model with respect to treatment of software development risks. The phases of the waterfall model do not include any reference to risk management. Risk Management is considered an ongoing activity that is part of the project management activities. Development progresses in a spiral shape consisting of multiple loops, where each loop represents a complete development cycle. The number of loops depends on project size, complexity, and risk.



2.2.1 Risk Handling in Spiral Model

- Risks are identified after development starts.
- Prototypes help detect and resolve risks early.
- Continuous risk evaluation at every iteration.
- More flexible than Prototyping and Waterfall models.

2.2.2 Phases of the Spiral Model

Each phase in this model is split into four sectors (or quadrants) as shown in Figure. In the first quadrant, some features of the product are identified based on the severity of the risk and how crucial it is to the overall product development. Implementation of the identified features forms the objective of the phase. The objectives are investigated, elaborated, and analyzed. Based on this, the risks involved in meeting the phase objectives are identified. Also, alternative solutions possible for the phase under consideration are proposed. During the second quadrant, the alternative solutions are evaluated to select the best possible solution. To be able to do this, the solutions are evaluated by developing an appropriate prototype. Activities during the third quadrant consist of developing and verifying the next level of the product. At the end of the third iteration, the identified features have been implemented and the next version of the product is available. Activities during the fourth quadrant concern reviewing the results of the stages traversed so far (i.e., the next version of the product) with the customer and planning the next iteration around the spiral. To make the model more efficient, the different features of the product that can be developed simultaneously through parallel cycles can also be identified. To keep our discussion simple, we shall not discuss about parallel cycles in the model. With each iteration around the spiral (beginning at the centre and moving outwards), progressively more complete versions of the software get built. The radius of the spiral at any point would represent the cost incurred in the project so far, and the angular dimension would represent the progress made so far in the current phase. In the spiral model of development, the project manager dynamically determines the number of phases as the project progresses. Therefore, while using this model, the project manager needs to play a very crucial role in tuning the spiral model to the specific project.

2.2.3 Pros of the Spiral Model

- Identifies and resolves risks at every phase, making it ideal for projects with high uncertainty.
- Allows changes in requirements even at later stages of development.
- Customers can review early versions of the product, improving satisfaction.
- Continuous testing and refinement lead to reliable and high-quality software.

2.2.4 Cons of the Spiral Model

- The model is complex due to multiple iterations and continuous risk analysis.
- Not suitable for small projects because it requires more time, effort, and resources.
- The number of iterations is not fixed, making scheduling difficult.
- Frequent reviews and evaluations can increase development time.
- Requires significant investment in planning, risk management, and testing.

III. AGILE DEVELOPMENT

In 2001, Kent Beck and 16 other noted software developers, writers, and consultants signed the "Manifesto for agile Software Development." It stated:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working-software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following plan

Agile software engineering combines a philosophy and a set of development guidelines. The philosophy encourages customer satisfaction and early incremental delivery of software; small, highly motivated project teams; informal methods; minimal software engineering work products; and overall development simplicity. The development guidelines stress delivery over analysis and design (although these activities are not discouraged), and active and continuous communication between developers and customers.Agile is used because it helps teams deliver value quickly and continuously. By prioritizing the delivery of difficult results early in the project,

customers benefit from seeing and using the product sooner, allowing for quick feedback and adjustments. Agile also encourages teams to focus on what truly matters, concentrating on tasks that add value and avoiding unnecessary work.The modern business environment that spawns computer-based systems and software products is fast-paced and ever changing. Agile software engineering represents a reasonable alternative to conventional software engineering for certain classes of software and certain types of software projects. It has been demonstrated to deliver successful systems quickly.

3.1 What is Agility?

Ivar Jacobson provides a useful discussion about Agility in the context of Software Engineering:

Agility has become today's buzzword when describing a modern software process. Every-one is agile. An agile team is a nimble team able to appropriately respond to changes. Change is what software development is very much about. Changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that inay have an impact on the product they build or the project that creates the product Support changes should be built-in everything we do in software, something we embrace because it is the heart and soul of software. An agile team recognizes that software is developed by individuals working in teams and that the skills of these people, their ability to collaborate is at the core for the success of the project.

In Jacobson's view, the pervasiveness of change is the primary driver for agility. Soft-ware engineers must be quick on their feet if they are to accommodate the rapid changes that Jacobson describes. But agility is more than an effective response to change. It also encompasses the philosophy espoused in the manifesto noted at the beginning of this chapter. It encourages team structures and attitudes that make communication (among team members, between technologists and business people, between software engineers and their managers) more facile. It emphasizes rapid delivery of operational soft-ware and de-emphasizes the importance of intermediate work products (not always a good thing), it adopts the customer as a part of the development team and works to eliminate the "us and them" attitude that continues to pervade many software projects; it recognizes that planning in an uncertain world has its limits and that a project plan must be flexible.

Agility can be applied to any software process. However, to accomplish this, it is essential that the process be designed in a way that allows the project team to adapt tasks and to streamline them, conduct planning in a way that understands the fluid-ity of an agile development approach, eliminate all but the most essential work prod-ucts and keep them lean, and emphasize an incremental delivery strategy that gets working software to the customer as rapidly as feasible for the product type and operational environment.

3.2 Agile Methodology:

Agile Methodology is a way to manage projects by breaking them into smaller parts. It focuses on working together and making constant improvements. Teams plan, work on the project, and then review in a repeating cycle.

- It focuses on delivering smaller pieces of work regularly instead of one big launch.
- This allows teams to adapt to changes quickly and provide customer value faster.
- Major companies like Facebook, Google, and Amazon use Agile because of its adaptability and customer-focused approach.
- Prioritize flexibility, collaboration, and customer satisfaction.

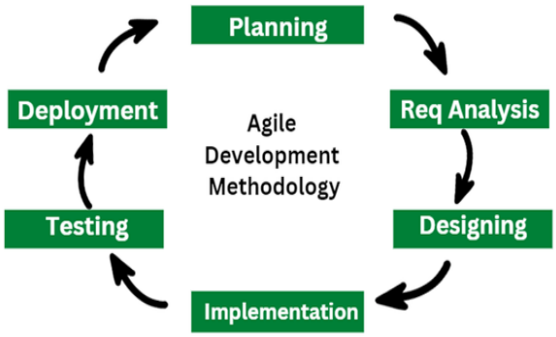
3.2.1 Agile Methodology Principles:

The Agile Alliance defines 12 agility principles for those who want to achieve agility:

1. Our highest priority is to satisfy the client through early and continuous delivery of valuable computer software.
2. Agile processes welcome changing requirements, even late in development, and leverage change to provide a competitive advantage for the customer.
3. Deliver operating computer software often, from a pair of weeks to a couple of months, with a preference to the shorter timescale.
4. Business individuals and developers should work together daily throughout the project.
5. The build comes around to activate people. offer them the setting and support they have, and trust them to complete the task.

6. The foremost economical and effective methodology of conveying information to and among a development team is face-to-face speech.
7. Working with computer software is the primary life of progress.
8. Agile processes promote property development. The sponsors, developers, and users will be able to maintain a relentless pace indefinitely.
9. Continuous attention to technical excellence and smart style enhances nimbleness.
10. Simplicity—the art of maximizing the number of works not done—is essential.
11. The most effective architectures, necessities, and styles emerge from self–organizing groups.
12. At regular intervals, the team reflects on a way to become simpler, then tunes and adjusts its behavior accordingly.

3.2.2 Agile Development Stages the Agile (SDLC) breaks a project into six structured stages, enabling continuous improvement, faster delivery, and better adaptability to change.



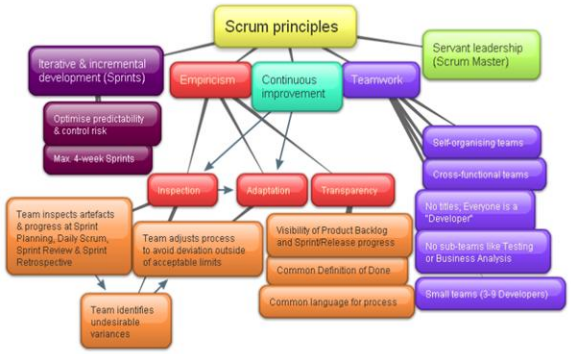
3.2.3 Types of Agile Methodology

Agile is a flexible framework with several approaches, each suited for different project needs. Below are some of the most common Agile methodologies, formatted for clarity:

- Kanban: It is an approach to evolutionary and incremental process improvement using a work-in-progress (WIP) limited pull system. It helps teams handle frequently changing priorities, continuous incoming work, and enables faster, frequent releases.
- Scrum: It works best for small teams and follows short, time-boxed sprints to deliver focused outcomes. A Scrum Master removes obstacles, while events like Sprint Planning and Sprint Retrospective help plan work and improve processes.

Scrum is most popular agile framework in the world, Scrum uses iterative and incremental development model. Scrum concentrates particularly on how to manage tasks within a team-based development environment. Scrum provides the simple framework of basic tenets to solve problems and deliver good results - more valuable software faster. Scrum beautifully defined in below diagram, that shows that scrum is based on following principles,

1) Iterative and incremental development
2) Continuous improvement
3) Empiricism (A theory that stats that knowledge comes only from experience)
4) Servant leadership (Scrum Master)



- Extreme Programming (XP): It focuses on strong technical practices and rapid delivery with continuous feedback. It emphasizes communication, simplicity, and frequent releases, making it ideal for projects with fast-changing requirements.
- Adaptive Project Framework (APF): It is designed for projects where requirements and conditions can change unexpectedly. It supports adapting to variations in resources, timelines, budgets, and team structures during execution.
- Extreme Project Management (XPM): It is suitable for highly complex and uncertain projects. It relies on short iterations and constant adjustments to processes and strategies as new challenges emerge.
- Adaptive Software Development (ASD): It focuses on continuous learning through the phases of Speculate, Collaborate, and Learn. The model is flexible, allowing teams to adapt quickly based on feedback and evolving project needs.

- Dynamic Systems Development Method (DSDM): It provides a structured Agile approach covering the full project lifecycle. It balances flexibility with control through phases like feasibility, iterative development, and final implementation.
- Feature Driven Development (FDD): It centers around delivering specific, client-valued features. It promotes frequent progress, quick fixes, and continuous feedback by developing and delivering features incrementally.3.2.4 Pros of Agile
- Agile places a high priority on providing customers with value by attending to their requirements and preferences. Agile guarantees that the most important features are produced first and that iterative changes are driven by customer feedback by dividing work down into small, manageable tasks.
- Agile gives teams the freedom to own their work and decide together. Team members feel motivated, proud, and owned when they have this autonomy together with a focus on providing value and ongoing growth.
- Throughout the development process, agile promotes strong coordination between product owners, developers, and other stakeholders. Better communication, a common understanding of the objectives, and ongoing feedback are all fostered by this partnership, which produces results that are higher quality and boost stakeholder satisfaction.
- Agile encourages the tiny, incremental releases of functional software. This gives early access to observable progress and facilitates early input and validation for stakeholders. Continuous delivery reduces risks by spotting problems early on and taking action to fix them.
- It is a key component of agile development, and this is emphasized by techniques like continuous integration, automated testing, and frequent inspection and modification. Agile guarantees that the software satisfies the required standards and lowers the likelihood of faults by integrating quality assurance throughout the development process.

3.2.5 Cons of Agile

- Project timeframes and outcomes might be difficult to predict with accuracy due to Agile iterative and incremental methodology. Stakeholders who need set budgets or timeframes may find this unpredictability troublesome.
- Agile highly depends on ongoing customer and stakeholder feedback and participation. Customers who are unavailable or who don't know enough about the domain can impede development and slow it down.
- While Agile works effectively for small to medium-sized teams working on relatively basic projects, scaling Agile methods to bigger teams or more complicated projects can be more difficult. As the project grows, it gets harder to maintain coordination, alignment, and communication.
- Agile focus on self-organizing, cross-functional teams with the authority to reach decisions together is paramount. Inadequate communication within the team or a lack of experience or expertise among team members can negatively affect output quality and productivity.
- Planning, coordinating, and communicating take more time and effort when using agile frameworks like Scrum. This overhead can take a lot of time, especially for projects with short deadlines or small teams.

3.2.6 Limitations of Agile

- Agile methodologies focus on less documentation; it prioritizes working on projects rather than paperwork.
- Busy schedule of clients can make daily meetup and face-to-face communication difficult.
- It may require experienced programmers to make critical decisions during the development of software.
- It has less rigid scope control, which may not be suitable in certain situations.
- Compared to more structured project management methods, it may lack predictability.

IV. COMPARISION BETWEEN WATERFALL MODEL AND AGILE METHODOLOGY

Agile and Waterfall are two popular methods for managing collaborative projects, but they have fundamentally different approaches. The core distinction lies in their structure. Agile is incremental and iterative, meaning it breaks projects into small, repeating cycles. In contrast, Waterfall is linear and

sequential, progressing through distinct phases one after the other.

These different approaches also lead to varied team management structures. In the Waterfall method, each project phase must be completed and approved before the next phase begins, with clear roles and a top-down approach. In contrast, Agile promotes continuous collaboration, adaptability, and cross-functional teams that work in short "sprints," allowing for flexibility and feedback throughout the entire project lifecycle.

Feature	Waterfall Methodology	Agile Methodology
Project Flow	You finish one stage before starting the next, like water flowing down a waterfall.	The project is broken into sprints. You build and improve incrementally.
Changes and Flexibility	Making changes once a stage is done is tough, costly, and can cause significant delays.	Welcomes new ideas or changes at any point. Flexibility is a primary goal.
Customer Involvement	Customers typically provide input mainly at the start and again at the very end, resulting in less ongoing feedback.	Customers are frequently involved, providing feedback after each small work cycle.
Paperwork	Detailed plans and documents are created at the beginning for every stage.	Focuses more on delivering working parts than on writing endless documents.
Handling Problems	Identifies and addresses all potential problems from the outset. Unexpected issues later are expensive.	Problems are identified and resolved in small, regular cycles, making it easier to adapt quickly.
Timeline and Delivery	The whole project's timeline and deliverables are usually set at the very beginning. You get the final product all at once at the end.	Timelines for small work cycles are fixed, but the overall project timeline can be adjusted. You get working pieces often, not just one big delivery at the end.
Teamwork	Teams often work in individual groups (design, coding, testing). Instructions usually come from the top down.	Teams often organize themselves and include people with diverse skills. They communicate every day.
Best For	Projects where you know exactly what you need from the start, and requirements won't change. Suitable for projects with strict guidelines or straightforward objectives.	Projects where ideas might change or aren't fully clear. Ideal for complex projects, those requiring frequent updates, or when you want close collaboration with your customers.

V. CONCLUSION

Agile principles provide a foundation for a flexible and efficient software development process. By emphasizing frequent delivery, embracing change, and fostering collaboration, Agile processes enable teams to adapt incrementally and sustain progress. Continuous improvement, technical excellence, and a sustainable work pace are key aspects of this methodology.

REFERENCES

[1] R. Mall, Fundamentals of Software Engineering, 3rd ed. New Delhi, India: PHI Learning Private Limited, 2009: This book explains core software engineering concepts, including traditional SDLC models such as Waterfall, Spiral, and Iterative models. It discusses project management, risk handling, and quality assurance, providing a strong theoretical foundation for comparing traditional and Agile development approaches.

[2] K. A. Saleh, Software Engineering, India ed. India: XYZ Publisher, Year: This book explains core software engineering concepts, including traditional SDLC models such as Waterfall, Spiral, and Iterative models. It discusses project management, risk handling, and quality assurance, providing a strong theoretical foundation for comparing traditional and Agile development approaches.

[3] R. S. Pressman, Software Engineering: A Practitioner’s Approach, 7th ed. New York, NY, USA: McGraw-Hill, 2010: This book explains

core software engineering concepts, including traditional SDLC models such as Waterfall, Spiral, and Iterative models. It discusses project management, risk handling, and quality assurance, providing a strong theoretical foundation for comparing traditional and Agile development approaches.

[4] IOSR Journal of Computer Engineering (IOSR-JCE), “Title of the Paper,” e-ISSN: 2278-0661, p-ISSN: 2278-8727, pp. 01–08: This book explains core software engineering concepts, including traditional SDLC models such as Waterfall, Spiral, and Iterative models. It discusses project management, risk handling, and quality assurance, providing a strong theoretical foundation for comparing traditional and Agile development approaches.

[5] Journal of Software Engineering & Software Testing, vol. 10, no. 2, May–Aug. 2025, ISSN: 2457-0516 (Online): This book explains core software engineering concepts, including traditional SDLC models such as Waterfall, Spiral, and Iterative models. It discusses project management, risk handling, and quality assurance, providing a strong theoretical foundation for comparing traditional and Agile development approaches.