Journal of Global Research in Electronics and Communication
Volume 1, No. 4, April 2025
Available Online at: https://jgrec.info/index.php/jgrec/index
ISSN: 2321-3175
RESEARCH PAPER

Advances in Software Development Life Cycle Models: Trends and Innovations for Modern Applications

Vikas Prajapati
Independent l Researcher
Prajapati.vikas2707@gmail.com

***Abstract*—A crucial framework for Effectiveness, quality, and timely delivery is ensured by the Software Development Life Cycle (SDLC), which is the methodical design of software programs. This study explores the evolution of SDLC models, focusing on traditional, Agile, and hybrid models, as well as developments like DevOps, cloud computing, continuous integration/continuous deployment (CI/CD), and microservices architecture that have completely changed the development process. The study emphasizes the necessity of flexibility, teamwork, and automation while highlighting the major developments and trends influencing the state of software development today. Conventional SDLC models, including the V-Model, Spiral Model, and Waterfall Model, have been widely utilized for structured and predictable software development. However, with the rise of dynamic market demands and fast-paced technological advancements, these traditional models have often fallen short of providing the flexibility and adaptability required for modern software projects. Agile methodologies emerged, promoting iterative development, stakeholder collaboration, and continuous delivery, making it highly suitable for contemporary software needs.**

***Keywords*—Software Development Life Cycle (SDLC), Agile Method, Waterfall Model, CI/CD, Hybrid Agile**

I. INTRODUCTION

The SDLC comprises a comprehensive strategy for the creation, modification, upkeep, and replacement of software systems. An overview of the software development approach is provided by the SDLC Model, which serves as a foundation for the overall process. The Waterfall model, V-shaped model, evolutionary prototyping model, spiral model, iterative and incremental model, and agile model are among the several types of models. To guarantee the project's success, it could be necessary to select the appropriate SDLC model based on the particular issues and specifications. There are benefits and drawbacks to each model[1]. It depends on how their initiatives are going, and they must use the model in accordance with the requirements. The life cycle of software development The SDLC is a methodical technique to finish the software development process on schedule while maintaining software quality [2]. Often referred to as the software development life cycle, the list of duties that must be fulfilled during the system development process is described in the system development life cycle.

In the field of information technology (IT), the systems development life cycle, in its various variations, continues to be one of the most traditional and often utilized approaches to software development and acquisition. It has changed throughout time in response to ever-evolving situations and paradigm shifts in the development or acquisition of software, but its core principles remain relevant today[3]. Although life-cycle phases have existed under many names and with varying numbers of steps, The SDLC is essentially sound in its established application in business, industry, and government. It has been said that, together with prototyping, the SDLC is one of the two most used approaches for developing systems nowadays. Therefore, it is still crucial for students now and tomorrow to understand the SDLC.

Software packages are produced by several organizations to furnish offices with amenities. Some issues arise in the initial phases of software development. In order to prevent these issues or challenges, software engineering and software programs are developed in an organized manner. The SDLC is a method that allows for minute-by-minute software development, increases the likelihood that the project will be completed on time, and guarantees that the end product complies with requirements [4]. The SDLC framework gives system designers and developers a set of steps to follow while creating software.

A project's alignment with stakeholder expectations and organizational goals is ensured by the choice of the SDLC model. These studies demonstrate how varied software projects are and how flexible and adaptive SDLC selection methods are necessary to meet changing project needs. The variety of software projects [5], each with its own set of needs, limitations, and stakeholder expectations, makes the choice of an SDLC model essential.

A software project consists of papers that explain how to use and operate the program, data structures that allow the application to effectively manipulate information, and instructions (computer programs) that, when run[6], give the necessary features, function, and performance. When beginning a new software project, software engineers must overcome a number of obstacles, such as creating methods for creating software projects that can readily handle diverse platforms and execution settings.

II. OVERVIEW OF SOFTWARE DEVELOPMENT LIFE CYCLE MODELS

In the current dynamic and competitive software development environment, the SDLC model selection is a crucial component in determining project success [5]. Development methods, resource allocation, and project timelines are just a few of the aspects that are impacted by the organized frameworks that SDLC models provide. It might be

difficult to select the best SDLC model for a given software project, though, because there are so many of them, ranging from traditional waterfall to modern agile approaches. There are several models for software development, and numerous organizations design and employ their own models. The model selection has a significant influence on testing. The independent phases are management, testing, and validation, and they are used at all levels. Every model has pros and cons, and the organization's demands must guide the choice. An outline of a software development process often has many steps that define the creation, replacement, maintenance, modification, and improvement of new software [7]. The software's quality and the new project's entire development phases are the main emphasis of the life cycle approach. The SDLC's objective is to create a high-quality program that satisfies clients' objectives and expectations and is completed within a specific time frame at a reasonable cost[8].

A. Traditional Software Development Life Cycle Models

A "software developer" is someone whose job it is to create and maintain computer programs, and this role has been there since the early days of electronic information processing (ENIAC) and vacuum tubes. In the many years since the first computer was invented, a lot has happened in the software development process. These methods have changed to take into consideration the latest advancements in development environments, computer power, and ideas on the organizational management of software development teams. In response to this growth, both public and commercial software development initiatives throughout the globe have produced innovative approaches to software development. Though they differ greatly in methodology, these approaches all aim to create software in the most cost-effective, efficient, and effective way possible.

1) Waterfall Model

The Waterfall model, one of the most traditional methods for software development life cycles shown in Figure 1, rigorously follows a sequential sequence. It gradually descends. It is a part of the procedure for creating software. Everything must remain in its proper sequence; therefore, they can't alter it to meet client demands. These issues are a result of this. - There will be no profit, no use of time, and the customer will not be satisfied. Their requirements will remain unfulfilled.

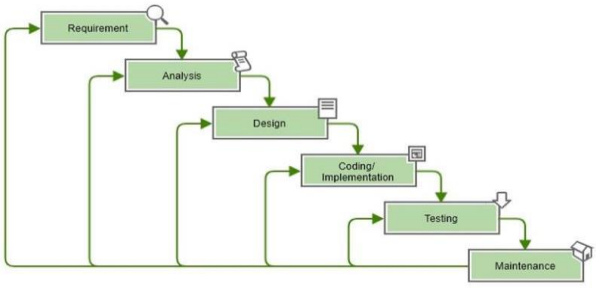

Fig. 1. Waterfall Model

2) V-Shaped Model

An expansion of the waterfall concept is the V-shaped shape, shown in Figure 2. The V-shaped model shows the relationships between the various stages of development and how they relate to testing. You may hear it called the "verification and validation model" as well. The most crucial aspect of every product is testing the software's functionality. A lot of testing is required [2]. The testing phase is thus the primary emphasis of the V-shaped model.

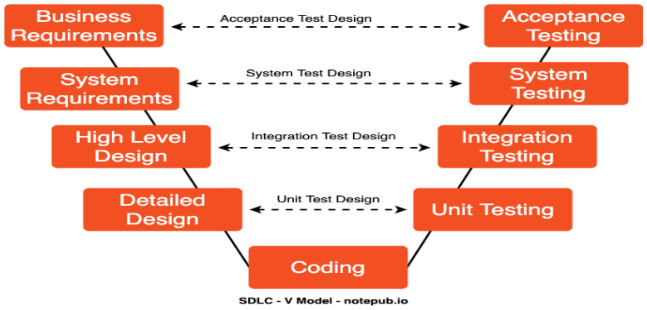

Fig. 2. V-Shaped Model

3) Iterative Model

One approach to the SDLC is the iterative model shown in Figure 3, which uses tiny units to apply software development requirements repeatedly until a final solution is reached. To accommodate software development process needs, the iterative or incremental approach is broken down into many architectural pieces. The four primary components of each architectural unit are design, development, testing, and implementation. According to the specifications of the suggested model, development proceeds iteratively until a final, publishable version is achieved.

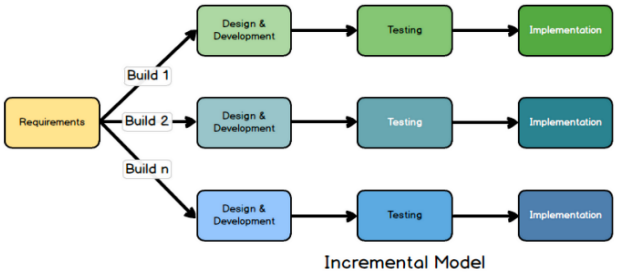

Fig. 3. Iterative Model

4) Spiral Model

The Spiral (waterfall) development model, which is based on risk analysis, is shown in Figure 4, and the iterative development models are combined in the four-stage spiral model [9]. In essence, the model iteratively improves the product by going around the coil.

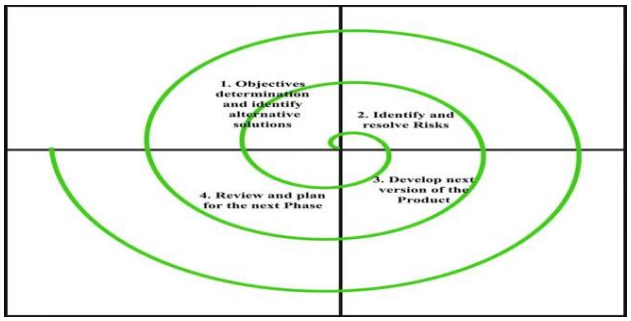

Fig. 4. Spiral Model

B. Agile Software Development Model

In agile development shows in Figure 5, this approach is useful for solving problems with data organization and structure, relational database architecture, and user interface by making use of process models to depict systems graphically. In order to accomplish iterative software enhancement, user feedback is utilized to converge on

solutions[10]. Agile development differs from traditional SDLC in that it breaks decomposing the SDLC into more manageable portions, referred known as "increments" or "iterations," that incorporate all of the conventional stages. Today, six approaches are recognized as agile development methods[11]. Software development methodologies such as crystal, feature-driven, lean, scrum, and extreme programming fall within this category.

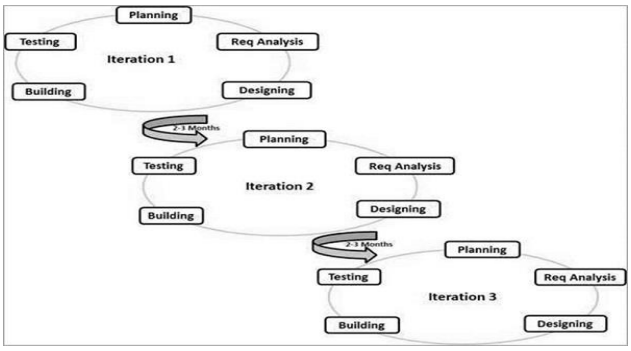

Fig. 5. Agile Model

C. System Development Life Cycle Model Classifications

There are three well-known ways to build systems: techniques that are structured, O-O analysis, and agile.

1) Structured Analysis

In the same way that a building's blueprint serves as an overarching plan for the building's construction, structured systems analysis is a tried-and-true, straightforward approach to systems development. This approach is useful for solving problems with data organization and structure, relational database architecture, and user interface by making use of process models to depict systems graphically.

2) Object-oriented Analysis

In terms of the nuts and bolts, object-oriented methods for building IT systems can make use of any of the tried-and-true techniques, such as waterfall, parallel, V-model, iterative, throwaway, and system prototyping. The RAD methodology for iterative development is commonly linked with object-oriented methods. Decomposition is the key differentiator between object-oriented methods and more conventional approaches like structured design. Conventional methods for solving decomposition problems focus on either processes or data.

3) Agile Methods

This is the most recent method that aims to build a system incrementally by simplifying, integrating the SDLC, and eliminating the need to spend time and energy on requirements creation and specification. Based on user input, agile methodologies typically utilize depicting a series of revisions or iterations through a spiral model[12]. A number of prototypes eventually give way to the final product as a consequence of the iterative process. Efficient administration of people, tasks, timelines, and budgets is essential in any development plan that makes use of project management tools and methodologies.

III. Emerging Trends in Software Development Life Cycle Models

The software development business is complicated and always evolving. Therefore, it's vital to have a deep understanding of all the approaches utilized in this field. To effectively handle uncertainty, one must have a thorough understanding of the many methods that are used. This is why it is standard practice to conduct thorough surveys and research studies to evaluate the current techniques' effectiveness and identify areas for improvement [13]. The software development team's credentials, the suggested solutions, the extent to which stakeholders are involved, and the possible financial effect are only a few of the many aspects considered in the quest for precise analysis [14]. Improving this relentless quest for advancement and improvement is crucial to the effectiveness and efficiency of software development processes.

A. Agile Software Development Method

A lightweight method of software development, the agile approach is frequently referred to as the "moving quickly" approach. Iterative or incremental software development is the foundation of this methodology, with each component standing in for a different phase of the project's evolution [9]. Agile methodology eschews conventional approaches like requirements analysis, planning, and tools in favor of client participation and engagement throughout the product development process. Many models make up this strategy, and we'll go over each one in depth.

Principles of Agile-

- Prioritize the timely and reliable delivery of high-quality software to ensure client satisfaction.
- Accept evolving needs, especially towards the end of development. the flexibility and adaptability offered by agile methods provide its customers a leg up in the market.
- Regularly release functioning software, ideally every two weeks to every two months, while a shorter timeframe is preferred[15].
- Developers and businesspeople must maintain constant communication and collaboration throughout the process.
- Build projects on persons who are highly driven. Encourage them, provide them with resources, and have faith in their abilities; they will succeed.
- The most efficient method of communication in a development team is face-to-face interaction.
- The most crucial measure of advancement is the presence of working software.
- Agile techniques help achieve sustainable growth. There ought to be no end to the number of users, sponsors, and developers who can keep up a steady pace.

Popular Frameworks of Agile Method-

1) Scrum

Scrum is yet another well-liked agile development technique that boosts productivity. Its foundation is essentially the incremental software development approach. The scrum technique refers to each iteration as a sprint, which divides the whole development cycle into a sequence of iterations. A sprint can last no more than thirty days[16]. The process begins with gathering user needs, albeit it is not anticipated that the user will provide all of them at the outset [17]. The needs must then be prioritized; this list is called the product backlog.

2) Extreme Programming (XP)

The concept of extreme programming was first developed by Kent Beck in 1996. Additionally, Ron Jeffries explains the process as the principles of Extreme Programming, a software development discipline, are boldness, communication, simplicity, and feedback. They provide people in customer, management, and programming positions with important rights and obligations. Unlike conventional techniques, XP is built on minor releases that are created on a periodic basis. It also prioritizes customer satisfaction alongside ongoing input. Therefore, it is important to embrace changes in specifications.

3) Crystal

Created by Alistair Cockburn in the early 1990s, the Crystal family of human-oriented, lightweight techniques aims to improve agility and efficiency. Crystal, according to Highsmith, prioritizes collaboration and teamwork by making use of project size, criticality, and objectives in order to establish appropriately configured practices for all members of the process family [18]. The following is a summary of Crystal's design principles: By using deeper lines of communication between individuals, the team can minimize intermediate work items as it creates running code more regularly [19].

B. DevOps Integration

DevOps came to be as a solution to the drawbacks of the present software development methodology and the gap between operations and development teams. The Waterfall model created silos between teams which delayed feedback, minimized collaboration, and resulted in misaligned objectives. Patrick Debois invented the term "DevOps" back in 2009 to understand the merging of development and operational processes in the hope of improving collaboration between those, reducing the time it took to bring software to the market, and enhancing the potential software's quality over time[20]. DevOps has been growing across multiple domains since it started. After a while, DevOps started to embrace practices like IaC, automated testing, and CI/CD, all address specific SDLC issues.

1) Continuous Integration (CI)

In software development, a technique known as continuous integration combines and builds code from each developer on a regular basis to identify defects as soon as feasible [21]. A CI system typically has the following steps, which are initiated by push or commit instructions.

- **Integrated-** At this point, every developer incorporates the scripts or output from their work.
- **Compiled Code**- is assembled into executables or packages.
- **Tested-** Test executables either automatically or manually.
- **Archived-** Archive logs, test results, and executable files while the procedure is underway.

2) Continuous Deployment (CD)

Software released using the CD methodology uses automated testing to verify that code base modifications are accurate and reliable enough for instantaneous autonomous deployment to a production environment [22]. Because of the terminology, it might be difficult to distinguish between continuous deployment and continuous delivery. Both have fairly identical tasks and are shortened to CD [23]. The first step in deployment is delivery. Prior to the production release in delivery, a last manual approval step is required.

3) Role of DevOps in SDLC

A major component of software development is DevOps. In software development, the traditional approach uses distinct teams for development, testing, and operations [24]. Each team has its own set of responsibilities and objectives as a result of this strategy's creation of a silo culture [25]. However, DevOps brings these teams together to work towards a shared objective.

IV. Innovations in Software Development Life Cycle For Modern Applications

The SDLC has seen significant advancements, driven by the need to build more complex, scalable, and adaptable applications in increasingly dynamic environments. Innovations in technology, tools, and methodologies have revolutionized how software is developed, tested, and deployed. Below are some key innovations in SDLC that are particularly impactful for modern applications:

A. Micro Services Architecture

The loose connectivity between different services is the most important factor in microservices design. The technical needs of the various services may vary, and they may share or possess their own servers and databases. Therefore, the design and implementation of microservices may vary depending on the application's scalability needs [26]. Appropriate communication and deployment strategies are also essential components of a dependable microservices architecture.

B. Cloud Computing And Software Development Life Cycle

The public, private, or hybrid cloud models and SLA distinguish the CSDLC from the typical software development life cycle[27]. The duties of cloud service providers and users, as well as business-level policies, are clearly outlined under SLA. The list of services, including SaaS[28], PaaS, IaaS, customization, security, accessibility, and multi-tenancy, is described in depth. The program may be modified by each user in the CSDLC, and the SLA contains information on customizations [29]. The CSDLC also requires data security and protection [30] to address both external and internal risks. Multiple users on various platforms can access the same program instance, thanks to multi-tenancy.

C. Artificial Intelligence And Automation

Software engineering jobs have been helped or mechanized by AI methods, which aim to build computer programs with an intellect comparable to that of a human being. There have been a lot of studies that have found bugs in software documents, such as requirements, designs, test plans, and source code, by using software inspections [31]. The area of study known as automated software engineering is always developing new approaches and tactics [32]. Mathematical model-based toolkits (theorem provers and model checkers), requirements-driven and reverse-engineering toolkits (design, coding, verification validation), project and configuration management tools, code generators, analyzers, and visualizers—all are part of this [33].

D. Continuous Integration/Continuous Deployment (CI/CD)

As software development has advanced, the conventional approaches of lengthy development result in major delays,

subpar products, and increased risks for major releases. The adoption of CI/CD procedures is a result of the demand for more effective and agile approaches. A culture of shared responsibility and cooperation is fostered by CI/CD, which bridges the gap between development and operations teams. [34]. Using CI/CD, teams are able to automate the build process, find and fix issues earlier, and test and deploy procedures to reduce human error and deliver code updates safely and promptly. This paradigm change increases software releases' dependability and speed while also improving their general quality and customer happiness.

V. Literature Review

Yusuf et al. (2023) Through improved team dynamics, practical advice, and theoretical knowledge, this study significantly advances the field of software development. Customer happiness, productivity, and project delays may all be decreased with the FCC approach. By exchanging research, specialists in the field may emphasize cooperation and teamwork, which will help software development methods continue to advance and improve. The SDLC has evolved to incorporate several models, such as the Waterfall, Spiral, V-Model, Iterative, Big Bang, and Agile cycles. The literature research reveals that the SDLC as a framework has six main problems and flaws: inadequate analysis, poor documentation structures, insufficient flexibility, difficulty choosing the appropriate SDLC, inappropriate design, and lack of adaptation[35].

Al Alamin & Uddin (2021) A thorough literature evaluation of a significant number of research publications devoted to ML model quality assurance is conducted in this work. They mapped the numerous ML adoption obstacles throughout the SDLC phases to create a taxonomy of MLSA quality assurance concerns. Based on the taxonomy, they offer suggestions and research possibilities to enhance SDLC procedures. This mapping can assist MLSAs in prioritizing their quality assurance initiatives when the use of ML models is deemed essential. Therefore, MLSAS must have quality assurance (QA). Numerous studies are devoted to identifying the particular difficulties that may arise when integrating ML models into software systems.[36].

Jayanthi et al. (2021) The significance of green software and how experts see it are highlighted in this study. It also discusses and explains the green software that is now available, as well as the strategies that organizations are using to make a more sustainable way to build software. They compare and contrast these methods. At the end of the study, the authors propose a system for organizations to follow that incorporates sustainability into software development at every level. As the first step in lowering their carbon impact, businesses must monitor their power management. Software greening is essential to business power management[37].

Aminu & Ogwueleka (2020) The purpose of this essay is to evaluate software life cycle models so that readers may choose the one that is most suitable for their clients. Accordingly, the developer chooses a model and attempts to meet the needs of the client. Software life cycle models are presented in this study along with a performance comparison. Every day, a new SDLC model is created in the software business, yet each model has advantages and disadvantages. Thus, none of them satisfy every need of the client[12].

Reyal et al. (2021) An interview and two polls are included in this paper: A review of existing literature on automatic UI-generating tools and a poll of users regarding the need for an automatic UI-generation system. The first study documents the current state of software and hardware solutions for creating user interfaces automatically. In order to close the gap between human and automated user interface creation, the second study adheres to the SDLC steps: requirements gathering, UML diagram generation, wireframe generation, web UI generation, evaluation of the application of HCI concepts, and HCI practices[38].

TABLE I. Literature of review on Advances in Software Development Life Cycle Models: Trends and Innovations for Modern Applications

Reference	Study On	Approach	Key Findings	Challenges	Limitations
Yusuf et al., (2023)	FCC model's impact on SDLC	Theoretical understanding and practical guidance for improved SDLC	FCC approach boosts customer happiness, productivity, and decreases project delays.	poor analysis, unclear SDLC choices, poor documentation, inappropriate design, and a lack of flexibility	Limited empirical validation of FCC model in diverse environments
Al Alamin & Uddin, (2021)	Quality assurance in ML-based SDLC	Literature review and taxonomy of ML adoption challenges	Developed taxonomy for MLSA quality assurance, prioritization of QA efforts	Ensuring reliability and accuracy of ML models in SDLC	Limited empirical studies on MLSA quality assurance
Jayanthi et al., (2021)	Green software in SDLC	Comparative analysis of energy-efficient SDLC methods	Discusses sustainability in software, the importance of power management	Adoption of green software development practices	Lack of universal standards for sustainable software development
Aminu & Ogwueleka, (2020)	Analysis of SDLC models in comparison	Comparison of many SDLC models' performances	No single SDLC model can satisfy every need of the client	Selecting an appropriate SDLC model for a project	The constant emergence of new SDLC models with varying effectiveness
Reyal et al., (2021)	Automatic UI generation in SDLC	Literature and user survey on UI automation tools	Analyzes gaps between manual and automated UI generation	Integrating HCI concepts with SDLC automation	Limited tools for full automation of UI generation

VI. Conclusion And Future Work

The SDLC models have undergone significant modification to accommodate the ever-changing requirements of modern software applications. Traditional SDLC models such as the Waterfall, V-model, and Spiral Model can often hinder fast expansion, particularly in settings that are constantly changing, even if they offer an organized approach, due to their lack of adaptability and flexibility. Agile approaches arose as a more flexible solution, with an emphasis on continuous delivery, stakeholder engagement, and iterative development. However, as these innovations evolve, new challenges arise, particularly in terms of quality assurance, security, and managing cross-functional teams in a highly dynamic environment. The continued advancement in automation, AI-driven testing, and machine learning within SDLC processes holds the potential to overcome some of these hurdles, providing more robust and intelligent solutions for modern software development. As the landscape of

software development continues to evolve, several areas warrant further exploration and development.

The future of SDLC models is exciting, with a shift toward more intelligent, automated, and scalable processes. To keep up with the rapid changes in technology and user needs, the industry must continue to innovate and adapt its development methodologies to meet the challenges of the future.

REFERENCES

[1] H. S. Chandu, "Advanced Methods for Verifying Memory Controllers in Modern Computing Systems," pp. 377–388, 2024, doi: 10.48175/IJARSCT-19862.

[2] G. Gurung, R. Shah, and D. Jaiswal, "Software Development Life Cycle Models-A Comparative Study," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, pp. 30–37, 2020, doi: 10.32628/CSEIT206410.

[3] M. Mcmurtrey, "A case Study of the a Pplication of the S Ystems D Evelopment L Ife C Ycle (Sdlc) in 21 St C Entury H Ealth C Are : S Omething O Ld, S Omething N Ew ?," pp. 1–12, 2022.

[4] B. Acharya and K. Sahu, "Software Development Life Cycle Models: A Review Paper," *Int. J. Adv. Res. Eng. Technol.*, vol. 11, no. 12, pp. 169–176, 2020, doi: 10.34218/IJARET.11.12.2020.019.

[5] D. B. Aniley, "Selection of Software Development Life Cycle Models using Machine Learning Approach," vol. 186, no. 42, pp. 36–43, 2024.

[6] A. Singh and P. J. Kaur, "Analysis of software development life cycle models," *Lect. Notes Electr. Eng.*, vol. 476, no. 2, pp. 689–699, 2019, doi: 10.1007/978-981-10-8234-4_55.

[7] A. Gogineni, "NOVEL SCHEDULING ALGORITHMS FOR EFFICIENT DEPLOYMENT OF MAPREDUCE APPLICATIONS IN HETEROGENEOUS COMPUTING," *Int. Res. J. Eng. Technol.*, vol. 4, no. 11, p. 6, 2017.

[8] A. Goyal, "Optimising Cloud-Based CI / CD Pipelines : Techniques for Rapid Software Deployment," *TIJER*, vol. 11, no. 11, pp. 896–904, 2024.

[9] A. Alazzawi, Q. Yas, and B. Rahmatullah, "A Comprehensive Review of Software Development Life Cycle methodologies: Pros, Cons, and Future Directions," *Iraqi J. Comput. Sci. Math.*, vol. 4, pp. 173–190, 2023, doi: 10.52866/ijcsm.2023.04.04.014.

[10] G. Modalavalasa, "The Role of DevOps in Streamlining Software Delivery : Key Practices for Seamless CI / CD," *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 1, no. 2, 2021, doi: 10.48175/IJARSCT-8978C.

[11] Y. Leau, W. K. Loo, W. Y. Tham, and S. F. Tan, "Software Development Life Cycle AGILE vs Traditional Approaches," vol. 37, no. Icint, pp. 162–167, 2012.

[12] H. Aminu and F. N. Ogwueleka, "A Comparative Study of System Development Life Cycle Models," *J. Emerg. Technol. Innov. Res.*, vol. 7, no. 8, pp. 200–212, 2020.

[13] E. Ailen OMIJIE, "Variations and Emerging Trends in Software Engineering," *Int. J. Sci. Res.*, vol. 13, no. 3, pp. 1675–1679, 2024, doi: 10.21275/sr24323044809.

[14] V. Pillai, "Enhancing Transparency and Understanding in AI Decision-Making Processes," *irejournals*, vol. 8, no. 1, p. 5, 2024.

[15] R. Shah, "A Literature Review on Agile Model Methodology in Software Development," vol. 3, no. 6, p. 2017, 2017.

[16] J. Thomas, "The Effect and Challenges of the Internet of Things (IoT) on the Management of Supply Chains," vol. 8, no. 3, pp. 874–878, 2021.

[17] S. Sharma, D. Sarkar, and D. Gupta, "Agile Processes and Methodologies: A Conceptual Study," *Int. J. Comput. Sci. Eng.*, vol. 4, 2012.

[18] B. J. Syed Mohammed Nadeem, Deepak Dasaratha Rao, Arpit Arora, Yashwant V Dongre, Rakesh Kumar Giri, "Design and Optimization of Adaptive Network Coding Algorithms for Wireless Networks," *IEEE*, pp. 1–5, 2024.

[19] I. Journal, "AGILE: Software development model 12 www.erpublication.org," no. 3, pp. 11–17, 2015.

[20] J. Thomas, "Enhancing Supply Chain Resilience Through Cloud-Based SCM and Advanced Machine Learning: A Case Study of Logistics," *J. Emerg. Technol. Innov. Res.*, vol. 8, no. 9, 2021.

[21] M. S. S Shah, "Kubernetes in the Cloud: A Guide to Observability," *DZone*, 2025.

[22] M. H. Rifa'i Istifarulah and R. Tiaharyadini, "DevOps, Continuous Integration and Continuous Deployment Methods for Software Deployment Automation," *JISA(Jurnal Inform. dan Sains)*, vol. 6, p. 116, 2023, doi: 10.31326/jisa.v6i2.1751.

[23] Abhishek Goyal, "Optimising Software Lifecycle Management through Predictive Maintenance: Insights and Best Practices," *Int. J. Sci. Res. Arch.*, vol. 7, no. 2, pp. 693–702, Dec. 2022, doi: 10.30574/ijsra.2022.7.2.0348.

[24] V. S. Thokala, "A Comparative Study of Data Integrity and Redundancy in Distributed Databases for Web Applications," *Int. J. Res. Anal. Rev.*, vol. 8, no. 4, pp. 383–389, 2021.

[25] S. Goel and B. T. Cse, "Role of DevOps in Full-Stack Web Development," vol. 8, no. 5, pp. 9–14, 2023.

[26] N. Torvekar and P. Game, "Microservices and Its Applications An Overview," *Int. J. Comput. Sci. Eng.*, vol. 7, pp. 803–809, 2019, doi: 10.26438/ijcse/v7i4.803809.

[27] M. S. Samarth Shah, "Deep Reinforcement Learning For Scalable Task Scheduling In Serverless Computing," *Int. Res. J. Mod. Eng. Technol. Sci.*, vol. 3, no. 12, pp. 1845–1852, 2021, doi: DOI : https://www.doi.org/10.56726/IRJMETS17782.

[28] S. Arora and S. R. Thota, "Automated Data Quality Assessment And Enhancement For Saas Based Data Applications," *J. Emerg. Technol. Innov. Res.*, vol. 11, pp. i207–i218, 2024, doi: 10.6084/m9.jetir.JETIR2406822.

[29] D. Jagli and S. Yeddu, "CloudSDLC: Cloud Software Development Life Cycle," *Int. J. Comput. Appl.*, 2017, doi: 10.5120/ijca2017914468.

[30] S. M. Adedapo Paul Aderemi Shomili Duary, Vandana Sharma, Pratyusha Choudhury, Deepak Dasaratha Rao, "Cybersecurity Threats Detection in Intelligent Networks using Predictive Analytics Approaches," *IEEE*, 2024.

[31] A. G. Milavkumar Shah, "Distributed Query Optimization for Petabyte-Scale Databases," *Int. J. Recent Innov. Trends Comput. Commun.*, vol. 10, no. 10, pp. 223–231, 2022.

[32] B. W. Sorte, P. P. Joshi, and V. S. Jagtap, "Use of Artificial Intelligence in Software Development Life Cycle: A state of the Art Review," 2015.

[33] Srinivas Murri, "Data Security Environments Challenges and Solutions in Big Data," vol. 12, no. 6, pp. 565–574, 2022.

[34] Y. Jani, "IMPLEMENTING CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT (CI/CD) IN MODERN SOFTWARE DEVELOPMENT," *Int. J. Sci. Res.*, vol. 12, pp. 2984–2987, 2023, doi: 10.21275/SR24716120535.

[35] M. Yusuf, M. K. Sophan, A. K. Darmawan, B. D. Satoto, D. R. Anamisa, and W. Agustiono, "Fast Collaboration Competencies Model for Software Development Life Cycle (SDLC)," in *2023 IEEE 9th Information Technology International Seminar (ITIS)*, 2023, pp. 1–6. doi: 10.1109/ITIS59651.2023.10420226.

[36] M. A. Al Alamin and G. Uddin, "Quality Assurance Challenges For Machine Learning Software Applications During Software Development Life Cycle Phases," in *2021 IEEE International Conference on Autonomous Systems (ICAS)*, 2021, pp. 1–5. doi: 10.1109/ICAS49788.2021.9551151.

[37] H. Jayanthi, A. Kulkarni, A. E. Patil, and S. S V, "An Organizational Structure for Sustainable Software Development," in *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, 2021, pp. 1539–1542. doi: 10.1109/ICAC3N53548.2021.9725722.

[38] S. Reyal *et al.*, "An Investigation into UI generation compliant with HCI standards ensuring artifact consistency across SDLC," in *2021 21st International Conference on Advances in ICT for Emerging Regions (ICter)*, 2021, pp. 93–98. doi: 10.1109/ICter53630.2021.9774787.

