Ms. Ritu Jangra, 2026, 14:4
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Technological Evolution in Software Development Life cycle: The Implications of Artificial Intelligence

Ms. Ritu Jangra[1], Riya Rani[2], Mayuresh Yadav[3]

[1,2]Computer Science and Engineering Department HMR Institute of Technology and Management Hamidpur, New Delhi-110036, Delhi,India

[3]AIML Department HMR Institute of Technology and Management Hamidpur, New Delhi-110036, Delhi,India,

Abstract- The Software Development Lifecycle (SDLC) is a pathway for software designing, modification, evaluation and optimisation. Baseline models like Waterfall and Agile go through problematic situations like inflexibility, delayed error detection, scope creep and obstacles in the management of complex projects. To tackle these anomalies, Artificial Intelligence (AI) took the spotlight by intensifying various phases through automation, predictive analytics, and smart decision-making. This review addresses the outcomes of AI in SDLC, outlining the improvement in required analysis, system designing, code generation and bugs detection as well as testing and deployment. AI technologies like GitHub Copilot and ChatGPT are reinforced to be the real-world applications that alter the developer productivity and software quality. AI serves various purposes like increased efficiency, reduced development times, higher accuracy and early bug detection. Despite these benefits, the study illustrates various demerits like data dependency and lack of transparency as well as ethical concerns. But it enhances the requirement of standardised frameworks, enriched explainability and optimised cooperation among humans and AI. Ultimately, AI can considerably boost the SDLC, but a balanced automation with human expertise is important to shape the future of software engineering by promoting more efficiency and steady and knowledgeable development in the processes.

Keywords: Software Development Lifecycle, Waterfall model, Agile Model.

I. INTRODUCTION

The building of a software product is subject to a construction known as the software development life cycle (SDLC). It is often seen as a part of the life cycle of systems development. Several models exist for these kinds of processes, each of which describes methods for a variety of tasks or activities that occur during the process [1]. The Software Development Life Cycle (SDLC), which includes stages such as requirement analysis, design, implementation, testing, and maintenance, provides an organized framework for producing high-quality software. It explains how software changes over time, from its original development to its operational stages and eventual retirement.

Additionally, a software process model functions as an abstract depiction of the software process's design or description [2]. Numerous SDLC models have been created, including the waterfall, Agile, and spiral models. Depending on the needs of the project, each model has special benefits and drawbacks. One phase must be finished before beginning the next in the waterfall model, which is described as a linear, sequential, non-iterative process. System feasibility, requirements analysis, design, coding and unit testing, system integration, installation, and maintenance are among the stages that this paradigm goes through. By allowing the results of one phase to be used as the input for the next, it attempts to ensure project success while addressing past software project challenges. [3].

Over time, the spiral model of the software process has developed, drawing on lessons learned from many iterations of the waterfall model applied to major government projects. Compared to preceding models, it provides a more solid foundation for software development, covering the majority of earlier models as special instances and offering advice on how to choose the best model combination for particular software scenarios [4]. An incremental and iterative methodology that prioritizes teamwork among self-organizing groups is known as agile software development.

It uses minimum ceremony to produce high-quality solutions quickly and effectively in response to stakeholder needs while operating within an efficient governance structure. It encompasses a number of techniques that use collaboration to develop requirements and solutions [5]. Traditional SDLC approaches, such as the Waterfall model, have serious problems despite advances in software development, including rigidity, delayed feedback, and high project failure rates. Although many of these problems are based on perceptions rather than scientific data, common challenges include difficulty adapting to change, late defect detection, and variable software quality owing to late testing. The Waterfall approach is still frequently employed in the software industry in spite of these difficulties. [6].

The Agile Manifesto serves as the foundation for a number of software development techniques used in agile methodology. According to the Scrum Alliance, Scrum consists of four artefacts (Product Backlog, Sprint Backlog, Product Increment, and Definition of Done), three primary roles (Product Owner, Scrum Master, and cross-functional team), and activities including Backlog Refinement and Sprint Planning. It does not, however, provide recommendations for scalability in larger projects. The Kanban approach, which relies on self-organized teams and leadership for successful Agile implementation, emphasizes concepts like work visualization and restricting work in progress despite having fewer required practices [7]. Software system development is expensive and difficult, with many success criteria but few post-mortems carried out to draw lessons from previous initiatives.

Many stakeholders put pressure on project managers and development teams, which has an impact on the software's price and quality. A mix of technical, project management, and business choices frequently leads to software project failures[8]. In software engineering, to manage all these bottlenecks Artificial Intelligence (AI) is becoming a game-changer, especially in software testing. The growing complexity of software is making traditional manual testing techniques insufficient. AI can improve accessibility, efficacy, and efficiency by automating and optimizing testing methods. Along with being in line with quick agile development cycles, it also tackles the lack of qualified testers. AI provides answers to a number of testing problems, such as the creation, optimization, and analysis of test cases[9]. Software engineering is changing as a result of recent developments in artificial intelligence, especially in deep learning architectures like transformer networks and massive language models. This change highlights the significance of generative AI and the need for sizable, objective datasets and benchmarks in order to train and assess AI systems. This paper aims to highlights all the pros and cons of traditional method and AI in SDLC field [10].

II. LITERATURE REVIEW

The Software development Life Cycle (SDLC) was described as a systematic framework that guided the creation of software products as mentioned by N Jain et al. [11] in their article . The author stated that various models existed to describe the various processes involved, with a lifecycle model encompassing a broad notion and specific software development methods aligning with it. SDLC was termed as the detailed planning for the creation, implementation, and eventual retirement of software, outlining logical steps along the process.

S Pargaonkar et al. [12] taken into account that SDLC served as a vital structure for describing the software development process, which included planning, design, implementation, testing, deployment, and maintenance as shown in Fig.1 . With the advancement of software engineering approaches, multiple SDLC models had arisen, each giving a unique approach to managing development.

Fig. 1 Software Development Life Cycle [12]

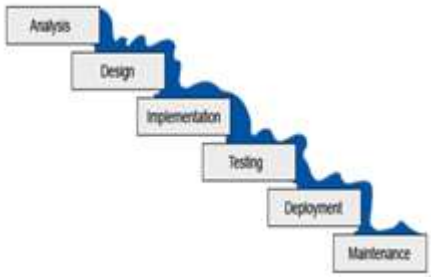
The SDLC alludes to many models for designing, developing, and producing high-quality, dependable, and cost-effective software products quickly. Major SDLC models included the Waterfall Model, Agile Model, Rapid Application Development Model, and Software Prototype, as mentioned by S. Shylesh et al. [13].

Software project development, a critical aspect of computing, fell under SDLC, aimed at risk mitigation and quality enhancement, highlighted by MA Rather et al. [14] in their paper. The SDLC provided a systematic framework to simplify and structure the software development process. It encompassed various activities and tasks essential for software creation.

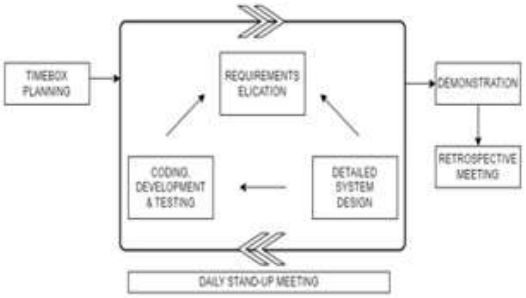
SM Salve et al. [15] spotlighted that high-quality software programs were designed, developed, and tested using an organized method called the Software Development Life Cycle (SDLC). It sought to guarantee that software fulfills client requirements, was finished on schedule, and stayed within budgetary constraints.

Enhanced human interface with software applications required software usability as visualized by S. Velmourougan et al. [16] in their research. Ignoring usability could lead to monetary loss, reputational harm, and a decline in trust. Since usability could not be sufficiently integrated at the conclusion of development, it must be prioritized at every stage of SDLC.

In recent decades, as stated by AF Diansyah et al. [17], many SDLC models had arisen to address the issues of the software industry. Key approaches such as Waterfall and Agile were evaluated based on flexibility, development speed, adaptability to changing needs, and project risk. The outcomes showed that each SDLC method had unique strengths and shortcomings that were appropriate for different settings. Waterfall provided structure and clarity, whereas Agile emphasized flexibility and teamwork as shown in Fig. 2.



(i) Waterfall model



(ii)Agile Model

Fig. 2 (i) Waterfall Model (ii) Agile Model [17]

S. Khan et al. [18] stressed out that the software development life cycle had two primary components: Process and Quality. Agile development encouraged iterative and adaptable techniques to meet client needs, whereas the waterfall model, often known as the linear sequential model, required the completion of one phase before moving on to the next. For better understanding, the comparison table is given below

Table 1. Comparison between Waterfall and Agile Method [18]

Aspect	**Water Fall Model**	**Agile Model**
Primary goals	Stability , high assurance	Rapid value , responding to change
Fundamental Hypothesis	It is Specific predicted and developed using extended and detailed planning	Based on fast feedback and changes, by small team, software that is of high quality with the use of continue improvement in design and

		testing is developed
User Requirements	Details and definition before Implementation	Interactively input
Communication method	Formal	Informal
Customer relations	Dedicated onsite customers, focused prioritized increments	Need Customer Interaction till the Contract
Quality control	Not easy to plan ,late testing and very difficult	Testing is done as the software is being developed, design, or requirement is permanent
Redeveloped cost	High	Low
Development direction	Fixed	Flexible
Project Size	Large project	Small or Medium size project
Testing	At the end of Coding	For each step

Agile and Waterfall approaches were used in software development, with emphasized on flexibility and customer collaboration. However, as projects expanded, typical agile techniques encounterd new issues in effort estimation, risk management, and backlog prioritization as accentuated by K. Ismail et al. [19] in their work. The evolution of artificial intelligence (AI) technologies provided solutions to these problems, resulting in an emerging field that improves project execution.

AI integration in Agile software development, as documented by B. Cabrero-Daniel et al. [20], improved manual procedures such as work allocation and backlog prioritization, resulting in faster product releases while preserving quality. Its popularity was especially evident in the automobile industry, where safety-critical software was produced using tools like test generators and user story conflict detectors.

PH Padmanaban et al. [21] foregrounded in their article that AI is a developing field that tackles a number of issues. It included techniques to improve software development activities, including learning-based systems, neural networks, fuzzy logic, and data mining. AI had enormous potential to improve every phase of SDLC as software develops in dynamic contexts.

Through automation, predictive modelling, and intelligent decision support, AI had improved productivity and product quality, as presented by SK Chintagupta et al. [22] in their research. The significance of AI models in software engineering was highlighted, with domain-specific models like Code BERT and GPT-based tools excelling in defect diagnosis and code generation. A visualisation in Fig. 3 aids in understanding the various AI techniques.

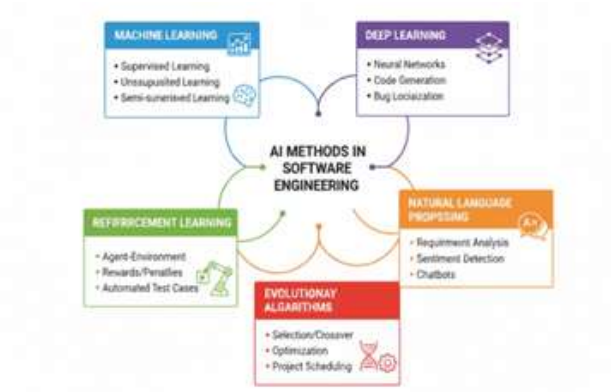


Fig. 3 AI Methods in Software Engineering [22]

L Arora et al.[23] suggested in their work that AI is gradually being integrated into daily life, yet its complicated and opaque models provide trust and acceptance issues, known as the black-box problem. Explainable Artificial Intelligence (XAI) aimed to increase AI systems' interpretability and transparency so that stakeholders can trust and respond to AI outcomes.

AI advances, particularly in Machine Learning (ML) and Deep Learning (DL), have resulted in their application in a variety of fields, including Software Engineering . S Cao et al. [24] pointed out that their black-box nature made practical implementation difficult, especially in key activities such as

vulnerability detection, where transparency in decision-making is essential.

T Wang et al. [25] presented that SDLC is critical for software quality, but understanding its ideas could be difficult due to the requirement for collaboration and access to a wide range of expertise. Dev Coach, a generative AI-powered system, assisted students in learning the SDLC by allowing them to interact with AI agents who simulate various roles in the software development team.

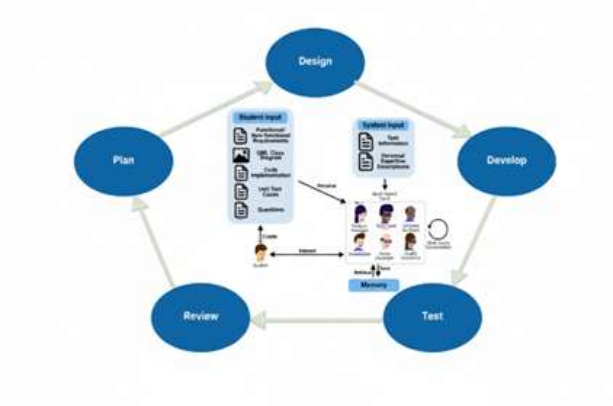


Fig. 4 Overview of the Dev Coach System: Students engage on assignments in various stages of the SDLC while learning and interacting with the multi-agent team driven by generative AI. In order to support students, the generative AI agents participate in multi-round conversations and interpret environmental information as input [25].

Generative AI (GenAI) is transforming software engineering, as specified by R. Gropler et al. [26], by generating code, discovering errors, recommending changes, and assisting with quality assurance. Its full potential across the SDLC remains untapped, with key uncertainties in reliability, accountability, security, and data privacy necessitating additional research. The emphasis is on GenAI's potential, new tool development, and emergent research problems that will define the future of software engineering.

The paper by E. Guimaraes et al. [27] suggested that the adoption of AI and large language models into software engineering is transforming the discipline by accelerating development and automating critical aspects of the software lifecycle. AI tools like OpenAI's Codex and DeepMind's AlphaCode improved productivity by understanding programming environments, predicting faults, and optimising solutions.

Software project management had evolved significantly in recent years as software systems have become more complicated, as stated by N. Anjum et al. [28]. Traditional project management approaches were frequently inadequate. AI, which included ML and NLP, improved project management by automating and providing predictive support, resulting in better decision-making, optimised resource allocation, and proactive risk management. AI adoption in software development resulted in accidental efficiency.

The suggested framework includes mandated training, human monitoring, and organizational rules to promote intentional sustainability as illustrated by FF Mohamed Iqbal et al. [29]. It recommended practical checkpoints for enterprises to incorporate into their SDLC processes, bridge the theory-practice gap for researchers, and underlines the necessity for developer-focused sustainability tools as well as the long-term impact of AI on technical debt.

Fairness management techniques throughout the SDLC address issues like model training, bias mitigation, and monitoring. Common fairness violations stem from data representation bias, algorithmic design bias, human judgement issues, and gaps in evaluation. Such violations could lead to serious repercussions, such as societal harm, privacy concerns, and diminished trust in AI decisions taken into account by T Nguyen et al. [30].

D. Ajiga et al. [31] spotlighted that the incorporation of AI into software development techniques in high-tech organizations was expected to result in dramatic shifts, driven by rising trends and technological developments. Notably, advancements in generative models improve skills not only in code generation but also in understanding and designing complicated code structures and designs, ultimately altering the industry and tackling existing difficulties.

III. CONCLUSION

Through the entire review it has been concluded that Artificial Intelligence (AI) has appeared to be the drastic force in the Software Development Lifecycle (SDLC), substantially improving efficiency, accuracy and automation among all phases of software development. However, traditional models such as Waterfall and Agile were substantial in structured development, while suffering from cons like rigidity, delayed error detection, scope creep and problematic management of large-scale projects. Whereas, consolidation of AI resolves many of these constraints by facilitating early bug detection, smart requirement analysis, automated testing and enhanced decision-making through statistical forecasting. Moreover, AI-powered methodologies such as ChatGPT, Copilot and GitHub demonstrate the actual working of AI in recent software engineering by modifying developer productivity and minimizing human efforts.

In spite of its various benefits, AI raises new issues such as over-reliance on automated systems, lack of transparency, data dependency and ethical concerns. These issues give the clear description that AI can't replace human skilled insights totally and should only be treated as an augmenting technology. In addition to this, the study explored various areas of enhancement, including the need for better integration techniques, standardized frameworks, greater security, privacy mechanisms and increased explainability. Dealing with these obstacles is important for fully confronting AI's potential in the software field. Eventually, AI is not exclusively an improvement but a paradigm shift in SDLC. The future of software engineering lies in the collaboration of humans and AI, where intelligent systems elevate human capabilities to achieve more robust, efficient and high-quality software solutions.

REFERENCE

1. Bhuvaneswari, T., and S. Prabaharan. "A survey on software development life cycle models." International Journal of Computer Science and Mobile Computing 2, no. 5 (2013): 262-267.
2. Kaur, Rupinder, and Jyotsna Sengupta. "Software process models and analysis on failure of software development projects." arXiv preprint arXiv:1306.1068 (2013).
3. Khan, Mohd Ehmer, S. G. M. Shadab, and Farmeena Khan. "Empirical study of software development life cycle and its various models." International Journal of Software Engineering 8 (2020): 16-26.
4. Boehm, Barry. "A spiral model of software development and enhancement." ACM SIGSOFT Software engineering notes 11, no. 4 (1986): 14-24.
5. Moniruzzaman, A. B. M., and Dr Syed Akhter Hossain. "Comparative Study on Agile software development methodologies." arXiv preprint arXiv:1307.3356 (2013).
6. Petersen, Kai, Claes Wohlin, and Dejan Baca. "The waterfall model in large-scale development." In International Conference on Product-Focused Software Process Improvement, pp. 386-400. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.
7. Alqudah, Mashal, and Rozilawati Razali. "A review of scaling agile methods in large software development." International Journal on Advanced Science, Engineering and Information Technology 6, no. 6 (2016): 828-837.
8. Cerpa, Narciso, and June M. Verner. "Why did your project fail?." Communications of the ACM 52, no. 12 (2009): 130-134.
9. Islam, Mahmudul, Farhan Khan, Sabrina Alam, and Mahady Hasan. "Artificial intelligence in software testing: A systematic review." In TENCON 2023-2023 IEEE Region 10 Conference (TENCON), pp. 524-529. IEEE, 2023.
10. Ahmed, Iftekhar, Aldeida Aleti, Haipeng Cai, Alexander Chatzigeorgiou, Pinjia He, Xing Hu, Mauro Pezzè, Denys Poshyvanyk, and Xin Xia. "Artificial intelligence for software engineering: The journey so far and the road ahead." ACM Transactions on Software Engineering and Methodology 34, no. 5 (2025): 1-27.
11. Jain, Namrata, and Anurag Jain. "Software Development Life Cycle: A Detailed Study." International journal of advanced research in computer science 2, no. 3 (2011).

12. Pargaonkar, Shravan. "A comprehensive research analysis of software development life cycle (SDLC) agile & waterfall model advantages, disadvantages, and application suitability in software quality engineering." International Journal of Scientific and Research Publications 13, no. 8 (2023): 120-124.
13. Shylesh, S. "A study of software development life cycle process models." In National Conference on Reinventing Opportunities in Management, IT, and Social Sciences, pp. 534-541. 2017.
14. Rather, Manzoor Ahmad, and Mr Vivek Bhatnagar. "A comparative study of software development life cycle models." International Journal of Application or Innovation in Engineering & Management (IJAIEM) 4, no. 10 (2015): 23-29.
15. Salve, S. Madhukar, S. Neha Samreen, and Neha Khatri-Valmik. "A comparative study on software development life cycle models." International Research Journal of Engineering and Technology (IRJET) 5, no. 2 (2018): 696-700.
16. Velmourougan, Suburayan, P. Dhavachelvan, R. Baskaran, and B. Ravikumar. "Software development Life cycle model to build software applications with usability." In 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 271-276. IEEE, 2014.
17. Diansyah, Ahmad Febri, Muhammad Rusdi Rahman, Rizky Handayani, D. Diffran Nur Cahyo, and Ema Utami. "Comparative analysis of software development lifecycle methods in software development: A systematic literature review." International Journal of Advances in Data and Information Systems 4, no. 2 (2023): 97-106.
18. Khan, Shamsulhuda, and Shubhangi Mahadik. "A comparative study of agile and waterfall software development methodologies." International Journal of Advanced Research in Science, Communication and Technology 2, no. 1 (2022): 399.
19. Ismail, Khaled. "Agile Practices and AI Integration: A Systematic Literature Review." Information and Software Technology (2026): 108119.
20. Cabrero-Daniel, Beatriz. "AI for Agile development: a Meta-Analysis." arXiv preprint arXiv:2305.08093 (2023).
21. Padmanaban, P. Harish, and Yogesh Kumar Sharma. "Implication of Artificial Intelligence in Software Development Life Cycle: A state of the art review." 2019 IJRRA all rights reserved (2019).
22. Chintagunta, Satyadhar Kumar. "The Role of Artificial Intelligence in Software Engineering: A Review of Frameworks, and Impact on the Software Development Life Cycle." International Journal of Emerging Research in Engineering and Technology 6, no. 4 (2025): 72-79.
23. Arora, Lakshit, Sanjay Surendranath Girija, Shashank Kapoor, Aman Raj, Dipen Pradhan, and Ankit Shetgaonkar. "Explainable artificial intelligence techniques for software development lifecycle: A phase-specific survey." In 2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 2281-2288. IEEE, 2025.
24. Cao, Sicong, Xiaobing Sun, Ratnadira Widyasari, David Lo, Xiaoxue Wu, Lili Bo, Jiale Zhang et al. "A Systematic Literature Review on Explainability for ML/DL-based Software Engineering." ACM Computing Surveys 58, no. 4 (2025): 1-34.
25. Wang, Tianjia, Matthew Trimble, and Chris Brown. "DevCoach: Supporting Students Learning the Software Development Life Cycle with a Generative AI powered Multi-Agent System." In Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, pp. 987-998. 2025.
26. Gröpler, Robin, Steffen Klepke, Jack Johns, Andreas Dreschinski, Klaus Schmid, Benedikt Dornauer, Eray Tüzün et al. "The Future of Generative AI in Software Engineering: A Vision from Industry and Academia in the European GENIUS Project." In 2025 2nd IEEE/ACM International Conference on AI-powered Software (AIware), pp. 170-181. IEEE, 2025.
27. Guimaraes, Everton, and Nathalia Nascimento. "AI in the Software Development Lifecycle: Insights and Open Research Questions." In Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, pp. 1353-1357. 2025.

28. Anjum, Nirjhor, Md Anwarul Kabir, and Kazi Jahanul Islam. "A Comprehensive Review of AI-Driven Project Management Techniques in Software Development." International Advanced Research Journal in Science, Engineering and Technology (IARJSET) 12, no. 6 (2025): 269-275.
29. Mohamed Iqbal, Fathima Fazla. "Framework for sustainable integration of artificial intelligence in to software development life cycle: insights from state of art and state of practice." (2025).
30. Nguyen, Thanh, Chaima Boufaied, and Ronnie de Souza Santos. "A Gray Literature Study on Fairness Requirements in AI-enabled Software Engineering." arXiv preprint arXiv:2512.07990 (2025).
31. Ajiga, Daniel, Patrick Azuka Okeleke, Samuel Olaoluwa Folorunsho, and Chinedu Ezeigweneme. "Enhancing software development practices with AI insights in high-tech companies." Computer Science & IT Research Journal 5, no. 8 (2024): 1897-1919.