Journal of Global Research in Multidisciplinary Studies (JGRMS)
Volume 2 | Issue 8 | August 2026 | Pages 1-8
DOI: https://doi.org/10.67805/jgrms.v2i8.150
e-ISSN: 3049-2963
RESEARCH ARTICLE OPEN ACCESS
Available online: https://saanvipublications.com/journals/index.php/jgrms

Software Development Methodologies Evolutionary Trends from Waterfall to DevOps

Mr. Raman Kumar
Assistant Professor
Department of Computer Science and Applications
Mandsaur (M.P.)
raman.kumar@meu.edu.in
Corresponding Author: *Mr. Raman Kumar (raman.kumar@meu.edu.in)

|**Received:** 25 July 2026 | **Revised:** 2 August 2026 | **Accepted:** 5 August 2026 |

***Abstract*—Agile and DevOps are currently the most popular software development techniques. Software methodologies started with the application of the Waterfall model and other models of Software development. It provides a general overview of the approaches and their integration throughout the Software Development Life Cycle (SDLC). SDLC phases are described and different SDLC methodologies are discussed with respect to their impact on motivation, productivity and quality. A comparison of the traditional Waterfall and newer iterative/incremental/agile is identified, as are the reasons behind this change, such as customer participation, timely feedback, and quality improvement. The following issues are common to all methodologies: architecture limitations, scalability and organizational restrictions. It was identified that the industry tends to follow a hybrid Agile-DevOps practice with more emphasis on automation, continuous integration and collaboration to achieve quick and competitive software environments.**

***Keywords*—Software Development Life Cycle (SDLC), Software Development Methodologies, Waterfall Model, Agile Software Development, DevOps, and Iterative and Incremental Models.**

I. INTRODUCTION

The term "software engineering" describes a process that includes a number of treatments, including requirement analysis, design, coding, testing, and maintenance procedures, all of which support the production of labor. Methodologies that offer structured procedures to handle complexity and guarantee adherence to project objectives serve as some of the guidelines for these procedures. Since then methods have evolved as time has gone on based on users' preferences, organizational needs, and technological advances [1]. This evolution has given rise to a variety of models including Waterfall, and more flexible and iterative options like Agile and DevOps.

The software development life cycle consists of two parts: software and process quality and processes [2]. All of these parts contribute to making software. The agile software development process is dynamic and iterative—requirements can be adapted and changed to meet individual client needs [3]. It simplifies time boxing and iterative development as well as adaptive planning. It is a concept approach and enables predictable interactions during the development process [4].

The Software Development Life Cycle (SDLC) is one of the models used to pinpoint activities performed during each phase of a software application development program, which include waterfall, rapid software application development (RAD), and spiral. All software development activities such as planning, analysis, design, coding, testing, and maintenance should be based on the needs of the client [5]. This relies on the particular uses to choose the model.

This change is crucial in the modern cut-throat market where companies need to be more innovative and responsive to their clients' feedback than ever before [6][7]. The combination of Agile and DevOps can transform how development teams work, interact with each other and deliver software, improving the development process, product quality, and customer satisfaction.

Infrastructure as code, automated testing, continuous integration and delivery, monitoring and logging, and continuous integration and delivery are some of the key practices of DevOps [8]. These methods could help organizations deliver more software with fewer bugs, issues, and downtime, leading to higher customer satisfaction and profitability.

A. Structure of the Paper

The paper is organized is as follows: The SDLC, or Software Development Life Cycle, is outlined in Section II. Section III delves into the topic of traditional software development approaches. Section IV reflects on evolutionary trends and observations. A literature review is presented in Section V. Finally, but just as importantly, Section VI concludes the work and suggests future research directions.

II. OVERVIEW OF SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

SDLC aids in producing high-quality software in a timely manner while keeping costs down (see Fig 1). Though individual programs may have wildly different quality levels, some common metrics include:

J. Glob. Res. Multidiscip. Studies, 2 (8):01-08
DOI: https://doi.org/10.67805/jgrms.v2i8.150

- The reliability of the program's features
- Overall performance
- Security
- Ultimately, the user experience

Fig. 1. Phases of (SDLC)

The SDLC is a method that divides the creation of software into manageable, reusable steps. Phases of the SDLC are guided by their own objectives and deliverables [9]. The SDLC is a set of interconnected steps that, when followed, provide a road map for teams to follow while making software that satisfies stakeholders, project objectives, and consumer expectations [10]. There are a number of SDLC models, and they all take different approaches to the different SDLC phases. Some models, like the waterfall model, have the stages finished in a specific order. Agile and other iterative processes allow for the simultaneous completion of multiple phases.

A. *Phases of SDLC*

The seven Phases of SDLC:

1) *Project Planning*

"What do they want?" is an important question to ask at the beginning of the SDLC. In order to determine what features the new software must have and how much its cost, the software delivery lifecycle must include project planning [11].

2) *Gathering Requirements & Analysis*

The second stage of the SDLC involves gathering information from clients regarding their product requirements. Discuss with the consumer all of the product's features and aspects. The development team then considers the architecture and source code of the software while analyzing the requirements. The main goal of this stage is to make sure that everyone is fully aware of the demand.

3) *Design*

In the third stage of the SDLC, the design phase, the programmer makes sure the end product meets all user needs. He also takes the client's technological, practical, and financial requirements into account while assessing the project's viability[12]. After settling on a strategy for the software's architecture, the developer typically chooses among popular programming languages like Oracle, Java, etc.

4) *Coding or Implementation*

This fourth step of the SDLC involves breaking the tasks into smaller units or modules and assigning each developer one of those [13]. The next step in building the system is for the developers to start writing code in their preferred programming languages. The SDLC includes this lengthy but powerful phase. To put code into action, developers need coding standards, interpreters, compilers, and debuggers, among other tools.

5) *Testing*

The software is released into the testing environment once development is complete. After that, the testing crew looks over the system's functionality in general[14]. Testing is the fifth step of the SDLC, and it's purpose is to make sure the end product satisfies the client's needs.

6) *Deployment*

The sixth step in the software development life cycle is releasing the product to customers. This happens when all the testing is done, and the software is ready to be used [15]. The deployment's complexity is proportional to the project's size. The next step is for the users to get the necessary training or manuals to run the software. Again, production goes through a short testing process to find and fix any problems with the new release that could be harmful to the environment.

7) *Maintenance*

The client's use of the developed system presents the true test, and it's important to fix these problems on a regular basis. This seventh stage of the SDLC focuses on ensuring the continued viability of the produced product. This software is updated often to keep up with changes in the end user's surroundings or technology. Included in Table I.

TABLE I. THE SEVEN PHASES OF SDLC

Phase	Key activities	Deliverables
1. Planning	Identify project scope, goals and requirements	Initial project plan
2. Analysis	Gather and review data on project requirements	Fully detailed requirement documentation
3. Design	Define project architecture	Software design document (SDD)
4. Coding	Write initial code	Functional software prototype
5. Testing	Review code and eliminate bugs	Refined, optimized software
6. Deployment	Deploy code to the production environment	Software available to end users
7. Maintenance	Continual fixes and improvements	Updated and optimized code

B. *Role of Methodologies in SDLC Efficiency*

The waterfall model captures the essence of the SDLC by outlining the process phases in accordance with the logic required to execute them. The output of one step is used as the input of the next step [16]. In a typical workflow, the following steps would be completed in the following order, with the most crucial ones highlighted in parentheses: feasibility, analysis, design, coding and unit testing, testing, and implementation. The conventional result of a waterfall process is a fully functional application [17]. Agile, spiral, and prototype are some alternatives to waterfall that are iterative and not sequential. An alternative to the waterfall methodology that does not follow a strict sequential order involves automating some functionality at a time through iterative development. It can be shown in Table II.

J. Glob. Res. Multidiscip. Studies, 2 (8):01-08
DOI: https://doi.org/10.67805/jgrms.v2i8.150

TABLE II. PERSPECTIVES FROM LIFE CYCLES AND METHODOLOGIES

	Life Cycles	Methodologies	
Characteristic	Sequential SDLC	Iterative SDLC - Prototype, Agile	Soft Systems Methods -
Purpose	Design and implementation of work support systems	Focus on functionality and/or timing of delivery	Focus on contexts and stakeholder rights
Goal	Complete functionality	On time, short-term delivery of partial functionality	Contextualized design
Perspective	Design thought processes	This period's functionality	Organization, information, technology, and socio-technical aspects of the problem

There is a lack of a thorough set of criteria for assessing and thinking about things in relation to application development in any of these SDLC and methodology options; all they provide are tools and procedures [18]. The SDLCs and methodologies do not, on their own, provide any guidance on how to address the shortcomings of application development or enhance them to meet the demands of modern applications. The next part analyzes application development practice's successes and failures to identify the most important qualities of successful applications.

C. *Sdlc Challenges Across Different Methodologies*

Scalability, maintainability, and performance are three areas where software could suffer from inadequate architectural design. An inflexible system that is hard to change or expand can be the consequence of bad design choices made early in the project [19]. This could also raise technical liability and future maintenance expenses, reducing the adaptability of the system to varying needs.

1) *Mitigation Strategies*

- **Architectural Reviews:** Regular architectural check-up with experienced architects to find and correct design flaws in their work early.
- **Prototyping:** Create prototypes of design to test architecture decisions and solicit early design feedback.
- **Scalability Planning:** Make sure the design of the architecture is scalable, so that it can be expanded and changed later on.

2) *Best Practices for Effective Design*

- **Design Patterns:** Resolve frequent design issues and facilitate the reuse of effective tactics by making use of well-known design patterns.
- **Iterative Design:** Embrace an iterative design process that permits frequent input and improvement.
- **Collaboration and Communication:** Developers, designers, and business analysts may work together more quickly and communicate more effectively to clarify and agree upon design goals and limitations.

III. TRADITIONAL SOFTWARE DEVELOPMENT MODELS

Software development models lay forth the groundwork for planning and executing software delivery cycles and milestones during an application's lifetime. Although there are many distinct models for design and development, they all essentially consist of the following steps: requirements gathering, design, implementation (coding), and testing or verification [20]. The execution of these stages is where the two differ primarily. Variations on the classic waterfall model are the most common development models used today.

A methodology outlining the various steps and processes involved in developing software, the SDLC Model [21]. Agile, iterative/incremental, spiral, evolutionary prototyping, and waterfall models are only a few examples of the numerous types of models.

A. *Waterfall Model*

The initial methodology for creating software was the waterfall model. Life cycle model with a linear sequential component is another name for it. The model is easy to grasp and put into practice. Following the completion of one stage by another, the waterfall model proceeds to the next [22][23]. One of the first methods used in SDLC is the waterfall model. Their paths cross precisely once in this model.

The success of the project and the development of software products were greatly influenced by the prevalence of the first approach in the SDLC, the waterfall method (Fig 2.).

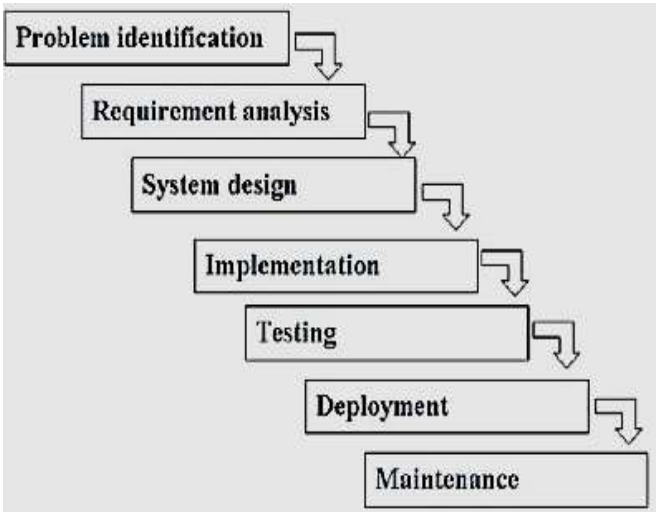

Fig. 2. waterfall model

The steps that make up the waterfall model in a logical order are:

1) *Requirement Analysis.*

This phase is crucial for gathering all the necessary requirements for the system development and documenting them in the software requirement specification document.

2) *System Design.*

The requirements from the previous phase are studied in this phase; these criteria help define the overall design and architecture of the system.

3) *Implementation.*

The fourth step is the specification of detailed and explicit requirements, which is followed by the development and construction of the software product in compliance with the specified standards. The software industry begins the process of product design and coding [24][25]. The creation of the software product's full functionality in accordance with the requirements validated by senior team members and all

J. Glob. Res. Multidiscip. Studies, 2 (8):01-08
DOI: https://doi.org/10.67805/jgrms.v2i8.150

project stakeholders is an example of a pre- and post-condition that is put into place during this phase.

4) Integration and Testing

Testing strategies are defined at this stage of the SDLC. It is at this step that modules are tested to ensure they are working and acting according to the requirements provided in the first phase or the standard definition.

5) Deployment

Once the product has been thoroughly tested and is officially ready for deployment, it is sent out in stages based on the company's deployment plans. At times, customer feedback also plays a role in product launch.

6) Maintenance

Software modification to suit customer needs and market trends is what this activity entails. To modify a system is to bring it up to modern standards so that it can efficiently satisfy the needs of the client and work with the technologies of today.

B. V-Model (Verification & Validation Model)

The foundation of this approach is the V-Model, which shows how each development step is associated with testing (Fig 3.). Software development and testing occur simultaneously because each step of development is accompanied by its testing phase in a V-shape.

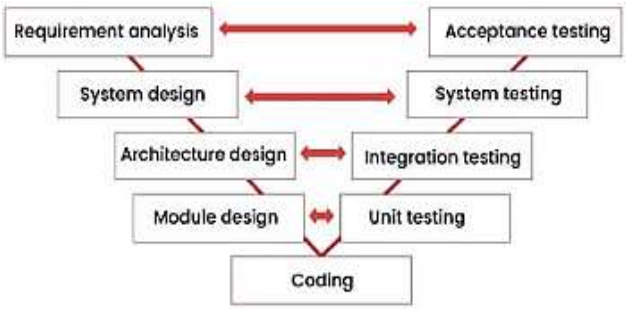

Fig. 3. V-model

1) Verification Phases:

- **Requirement Analysis:** In software development, requirements gathering is the first phase. During software development, requirements are to be satisfied, which include business needs [26]. In order to fully analyze an application's functionality from a user's point of view, business requirement analysis involves putting oneself in the customer's shoes.
- **System Design:** The process begins with the creation of a blueprint for the future system or application. The goal of system design is to create a comprehensive software and hardware specification.

The following are subcategories into which system design is further subdivided:

- **Architectural Design:** The primary focus of architectural design is the specification of technical approaches that used to accomplish software development goals. Architectural design, sometimes known as "high-level design," is concerned with outlining the big picture of a service, product, platform, or system.
- **Module Design:** The goal of module design, also called "low-level design," is to define the logic that used to build the system. They are now trying to show how the modules relate to one another and the order in which they interact.

2) Validation Phases:

- **Unit Testing Phase:** The purpose of unit testing is to find and repair bugs in specific modules. Running a small section of code to see if it performs as expected is all that is required for a unit test.
- **Integration Testing:** The phrase "integration testing" describes the process of combining different bits of code to ensure their seamless operation.
- **System Testing:** A system is ready to be tested once it has been fully assembled. Running the application in its intended environment allows us to determine if the system can efficiently perform with minimal response time.
- **User Acceptance Testing:** The requirement analysis phase is when the user acceptance test strategy is born, as the program needs to pass a battery of tests before it can be released into the wild to prove it has served its purpose.

C. Iterative and Incremental Models

Software development methodologies like the iterative model are commonly employed in Agile software development. This methodology breaks down software development into smaller, iterative cycles. A functional version of the program is created during each cycle, and then tested and feedback is collected. After that, it's business as usual until the software is finished. The iterative model's key benefit is that it facilitates quick feedback and development [27]. Because of this, software may be written and refined rapidly, with problems being found and fixed at an early stage.

Software development procedures that are characterized by incremental development are encompassed under the incremental model. The software's capability is enhanced with each increment. In most cases, the first increment is rather lengthy while the second one is pretty short. The main advantage of the incremental model is that the client can experience the product in its early stages. This allows for early comments to be collected and used to enhance the product going forward. By iteratively developing and testing new features before moving on to the next, the incremental model lowers project risk[28].

IV. Evolutionary Trends and Observations

The agile context has changed dramatically. There is a plethora of frameworks, techniques, and hybrid models for firms to navigate, rather than simply deciding between agile and waterfall approaches [29]. New possibilities and challenges have arisen for agile professionals as a result of the integration of new technologies such as ML, artificial intelligence, and automation tools. It has become necessary to modify the conventional agile frameworks due to the rise of remote and distributed teams brought about by global events.

A. From Waterfall to Agile: Drivers of Change

There are two popular project management frameworks that are used in the public sector: Waterfall and Agile. While each has its pros and cons, the choice of approach can significantly affect project delivery, particularly in a regulatory environment.

J. Glob. Res. Multidiscip. Studies, 2 (8):01-08
DOI: https://doi.org/10.67805/jgrms.v2i8.150

1) The Waterfall Methodology

The more traditional of the two methods is waterfall, which has long been the standard method in many industries, including the public sector. Waterfall was developed in the manufacturing and construction industry and the process is linear and sequential. Projects are broken up into different stages, and each stage relies on the successes or failures of the stage before it. Typically, the Waterfall methodology includes phases such as:

- **Requirements Gathering:** Stakeholders collaborate with project teams to set project goals, outcomes, and needs.
- **Design:** The requirements are used to develop detailed designs.
- **Implementation:** The product or service is produced by the project team based on the design specifications.
- **Verification:** The final step is to validate and test the product to make sure it meets all of the initial specifications.
- **Deployment:** The product is installed in the customer's / end user's environment. Maintenance Ongoing support and maintenance are provided as necessary after deployment.

2) The Agile Methodology

The Agile methodology differs from Waterfall with being iterative and flexible and has become a very popular approach in the last 20 years, especially in the software development and IT industry [30]. Agility, collaboration, and adaptability are the defining characteristics of agile. Project teams using Agile methodology are able to work in cycles, or "sprints," rather than in a sequential, phase-based fashion. These cycles enable the team to develop small, functional components of the project incrementally, with each sprint culminating in the delivery of a usable product or feature. Agile typically involves key processes such as: Initial

- **Planning:** Agile, in contrast to Waterfall, does not necessitate an initial stage with a precisely defined project scope [31]. A more effective approach would be to begin with a consensus on the project's broad objectives and then move on to iterative strategy sessions.
- **Development Cycles (Sprints):** Each sprint lasts a set amount of time (usually 2–4 weeks) and results in the delivery of a functional part of the product or solution.
- **Continuous Testing and Feedback:** The project is designed to incorporate testing and feedback at every stage. At the conclusion of each sprint, stakeholders offer feedback and participate in ongoing evaluations.
- **Adaptation and Iteration:** The product is refined over time as the project team adapts and iterates in response to stakeholder feedback and changing requirements.

B. Impact on Software Quality and Productivity

The idea of software engineering was born out of the so-called "software crisis"—a dire circumstance characterized by low productivity, poor quality, budget slippage, and inaccurate planning and cost estimation—in order to bring quality and productivity together.

The techniques involved in software development are distinct from those in conventional product manufacturing, and there are inherent differences between software and other goods as well [32]. One of the most domain-specific tasks that calls for a dedicated adaption guide to establish a universal quality standard is software development. One reason this hasn't happened is because software quality assurance techniques and methods can't just be copied from other industries' well-established quality practices (see Fig 4.). As a result, there has been a significant push to develop new and targeted models, processes, and methodologies for software development and to get reliable adaptations of activities for use in the development of other products.

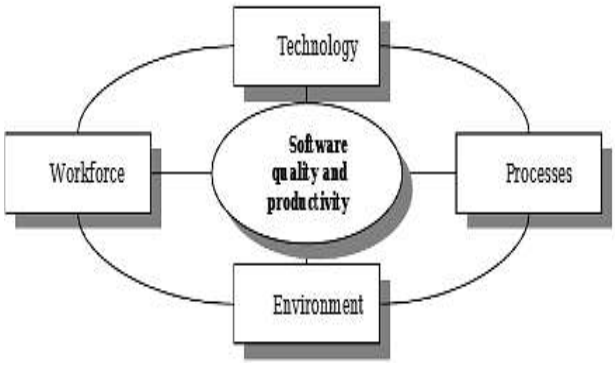

Fig. 4. Software quality and productivity

V. Literature Review

The literature study in Table III examined traditional software development approaches like Agile and DevOps. It highlighted the advantages of Agile, such as its iterative delivery and flexibility, and DevOps, which allows for rapid and quality software production by linking development and operations. The hybrid practices that embrace both Agile and DevOps solve the problems of collaboration and such organizational issues. The trends show a switch to DevOps-enabled Agile, emphasizing continuous integration, delivery, and improvement for the dynamic project environments.

Cao (2025) arranges and analyzes different models of software development in a methodical fashion; these models comprise the iteration-based, composite, flow-based, structured, and structured models. It also assesses the models' adaptability, risk management, skill requirements, and applicability to projects of varying sizes and contexts. Using eight critical criteria—flexibility, risk, time, expertise, project size, client engagement, delivery frequency, and quality assurance mechanisms—the literature review technique analyzes contemporary paradigms (such as Waterfall, Agile, and DevOps) [33].

Mumtaz et al. (2024) presented the software development life cycle and its history, as a backdrop for exploring various software development approaches. It thoroughly covers the traditional models such as Waterfall model and Iterative model, providing insights into their principles and applications. It also covers agile methodologies such as RAD and the Scrum model, exploring their suitability in today's software development landscape. The new DevOps approach is also thoroughly investigated, with an eye toward understanding its guiding principles, best practices, and the ways in which it combines both conventional and agile approaches [34].

Narang and Mittal (2023) The research explore and contrast the availability factor of important and selected

J. Glob. Res. Multidiscip. Studies, 2 (8):01-08
DOI: https://doi.org/10.67805/jgrms.v2i8.150

features in software development methods. The software firms benefited by choosing the appropriate method for the job. In Traditional and RAD approaches, the iterative, Waterfall, Prototype and Spiral development models are contrasted with those of Agile approaches: Scrum, Kanban and XP, as well as the DevOps automation culture and its key characteristics. Analysis is performed by studying different methodologies used in the development from current literature, Google and stack over flow trends [35].

Tsapa (2022) This blending is complex and has repercussions; it aims to face difficulties, propose solutions, integrate a critique of integration, impact, and breadth, and so on. Agile is iterative and user-centered, with DevOps practice for coordination of automation and continuous deliveries. But, there are obstacles to integration, including resistance to integration, tool chain incompatibility, and rolling resistance. Embracing automation allows organizations to tackle these difficulties head-on and pave the route to success in this industry. Implementing DevOps and Agile techniques can lead to faster time to market, higher quality products, greater innovation, and a more agile organization [36].

Al Masud et al. (2022) Software engineering has seen significant improvements with the advent of Agile and DevOps. Software quality and development time are both significantly enhanced by these techniques. The high skill requirements and lack of cooperation among software project's development, testing, and delivery sectors are two of the many obstacles that have been found in the practical application of DevOps and Agile [37].

Gokarna and Singh (2021) The SDLC is evolving to incorporate devOps practices. The term "DevOps" suggests a merging of the two departments responsible for developing and running a system. After then, the different phases of the development lifecycle are integrated. The DevOps approach can be seen as an evolution of the Agile technique, with certain improvements. The DevOps technique prioritizes security, feedback, continuous integration, delivery, improvement, and improvement as key components of software operations and development [38].

TABLE III. REVIEW OF SOFTWARE DEVELOPNT MODELS AND DEVOPS INTEGRATION

Authors (year)	Focus Area	Key Findings	Approaches	Objectives	Featherwork
Cao (2025)	Comparison of software development models	In terms of adaptability, consumer involvement, and incremental delivery, agile models (Scrum, XP) really shine.	Analyzed Waterfall, Agile, and DevOps along eight dimensions; reviewed literature on flow-based, structured, iteration-based, object-oriented, and composite models.	Categorize and compare models based on flexibility, risk, expertise, project size, delivery, and QA	Adoption guidance for dynamic projects
Mumtaz et al. (2024)	Historical and modern SDLC, traditional & agile methods	DevOps bridges development and operations, fostering collaboration and improving software delivery efficiency	Comprehensive examination of SDLC approaches	Explore development approaches, including traditional, agile, and DevOps	Further integration studies of DevOps with traditional and agile methodologies
Narang & Mittal (2023)	Feature availability in software development methodologies	Trend towards DevOps; comparative analysis of features in Traditional, RAD, Agile, and DevOps methods	Literature review, Google & Stack Overflow trends, tabular comparison	Aid industry in selecting an appropriate methodology based on required features	Tracking evolving market trends in DevOps adoption
Tsapa (2022)	Integration of Agile and DevOps	Agile + DevOps enhances speed, quality, innovation, and organizational agility; integration challenges include cultural resistance and tool incompatibility	Review & critique of integration	Examine challenges and benefits of blending Agile and DevOps	Solutions for smoother Agile-DevOps integration, overcoming organizational resistance
Al Masud et al. (2022)	Agile and DevOps hybrid methodologies	Agile + DevOps improves software quality and speed; limitations include a lack of collaboration and high skill requirements.	Literature review and proposed hybrid model	Bridge gaps between Agile and DevOps by integrating DevOps principles into Agile	Development of a hybrid DevOps-enabled Agile methodology
Gokarna & Singh (2021)	DevOps in the Software Development Life Cycle	DevOps enables CI/CD, continuous improvement, faster feedback, and security; extends Agile methodology.	Review of DevOps building blocks & models	Review DevOps practices, challenges, and future improvements	Suggestions to improve DevOps adoption and practices

VI. CONCLUSION AND FUTURE WORK

Evolutionary trends in software development methodologies, from traditional plan-driven models like Waterfall and the V-Model to the modern ones, which are Agile and DevOps-oriented. The analysis indicated that the early methodologies had the advantages of predictability, documentation, and sequential control, which were suitable for stable and well-defined requirements, but still had the downsides of inflexibility, limited scalability, and inability to cope with rapid change. On the contrary, the iterative and incremental approaches, especially Agile, brought in adaptability, continuous feedback, and stronger customer involvement that made the processes very responsive and the products very aligned with the users' needs. DevOps was also

a big player in this shift – much of the collaboration was a shift between dev and ops – and so there was a newfound visibility acquired through the lens of automation, CI/CD. Overall, the study indicates that there is no one right way to do it: software development is becoming more context dependent, and hybrid methods that are better aligned with the goals of the organization, project complexity, and technology needs are effective.

Future research could involve empirical testing of hybrid Agile-DevOps models across different domains of industry, especially safety critical systems, large systems etc. Furthermore, for the life cycle of software development practices, the role of automation, smart testing and decision-making support provided by AI is a promising path towards further enhancement of the quality, productivity and flexibility of software.

- **Funding:** This research received no external funding.
- **Institutional Review Board Statement:** Not applicable.
- **Informed Consent Statement:** Not applicable.
- **Data Availability Statement:** The data supporting the findings of this study are available from the corresponding author upon reasonable request.
- **Conflicts of Interest:** The author declares no conflict of interest.

How to cite this article:
R. Kumar, "Software Development Methodologies: Evolutionary Trends from Waterfall to DevOps," *Journal of Global Research in Multidisciplinary Studies*, vol. 2, no. 8, Aug. 2026. Doi: https://doi.org/10.67805/jgrms.v2i8.150

REFERENCES

[1] M. Z. Haider Ali, A. Arif, S. Z. Haider, M. Azam, M. Malik, and A. Hussain, "A Comprehensive Review Of Software Development Methodologies: Models, Mindset, And Misunderstandings," vol. 3, pp. 996–1029, 2025, doi: 10.5281/zenodo.16414601.

[2] S. Kothapalli, P. K. Gade, H. P. Kommineni, and A. Manikyala, *DevOps for Software Engineers*. 2024.

[3] V. K. R. K. Koppireddy, "Passwordless Authentication: A Modern Approach to Secure Access," *J. Inf. Syst. Eng. Manag.*, vol. 10, no. 63, 2025.

[4] R. K. Kanneganti, "Security and Compliance Issues in Cloud-Based Deployments of Content and Workflow Management Systems," *Eastasouth J. Inf. Syst. Comput. Sci.*, vol. 2, no. 01, pp. 131–138, Aug, 2024, doi: https://doi.org/10.58812/esiscs.v2i01.1122.

[5] S. Khan and S. Mahadik, "A Comparative Study of Agile and Waterfall Software Development Methodologies," *Int. J. Adv. Res. Sci. Commun. Technol.*, pp. 399–402, Jul. 2022, doi: 10.48175/IJARSCT-5696.

[6] E. Ok, B. William, J. Owen, B. Barnty, and M. Idowu, "The Symbiosis of DevOps and Agile: Transforming Software Development Practices," 2025.

[7] S. R. Somula, "Mitigating Estimation Challenges in Large-Scale Projects: An Integrated Framework for Enhanced Accuracy and Adaptability," *IPHO-Journal Adv. Res. Sci. Eng.*, vol. 3, no. 10, 2025, doi: 10.5281/zenodo.17462211.

[8] V. Božić, "DevOps -software development methodology," 2023.

[9] S. K. Malaraju and S. K. Madishetty, "AI-Augmented Compiler Optimization for Energyefficient Software Execution on Embedded Systems," in *2025 14th International Conference on System Modeling & Advancement in Research Trends (SMART)*, Moradabad, Uttar Pradesh, India: IEEE, 2025, pp. 1–6, November. doi: 10.1109/SMART66937.2025.11389313.

[10] M. Raza, "The Software Development Lifecycle (SDLC): An Introduction," *https://www.bmc.com*, 2020.

[11] L. Bennett, "Software Development Life Cycle (SDLC) Phases & Models," *https://www.guru99.com*, 2025.

[12] S. Irfan, "AI-Driven Credit Scoring: A Study of Machine Learning Algorithms for Enhanced Risk Assessment," in *2026 IEEE International Conference on AI Engineering and Innovations (AIEI)*, IEEE, Mar. 2026, pp. 1–6. doi: 10.1109/AIEI69164.2026.11496824.

[13] K. Jangiti, "Design and Validation of a Machine Identity Governance Framework for AI Agents in Multi-Cloud Environments," in *SoutheastCon 2026*, 2026, pp. 1–6. doi: 10.1109/SoutheastCon63549.2026.11476363.

[14] R. Azmeera, "The Effects of Increased Software Errors on Software Product Functionality," University of the Cumberlands, 2025.

[15] V. Chaturvedi, "Modern Software Development with Java, Spring Boot, and Python: A Survey of Frameworks and Best Practices," *ESP J. Eng. Technol. Adv.*, vol. 3, no. 4, pp. 188–197, December, 2023, doi: 10.56472/25832646/JETA-V3I8P121.

[16] M. Mittal, "The Great Migration: Understanding the Cloud Revolution in IT," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, vol. 10, no. 6, pp. 2222–2228, Dec. 2024, doi: 10.32628/CSEIT2410612423.

[17] S. Conger, "Software Development Life Cycles and Methodologies: Fixing the Old and Adopting the New.," *IJITSA*, vol. 4, pp. 1–22, 2011.

[18] J. B. Mehta, "Autonomous Workload Right-Sizing for Multi-Cloud Cost Optimization," in *2026 International Conference on Artificial Intelligence, Systems, and Emerging Technologies (ICAISET)*, Cairo, Egypt: IEEE, 2026, pp. 1–11, June. doi: 10.1109/ICAISET66439.2026.11541899.

[19] N. Gupta and M. Sahu, "Problem Faced During the Software Development Cycle," 2024. doi: 10.55041/IJSREM35471.

[20] L. N. Lior, "Design and Development Models and Processes," in *Writing for Interaction*, L. N. Lior, Ed., Boston: Elsevier, 2013, pp. 21–42. doi: 10.1016/B978-0-12-394813-7.00002-X.

[21] G. Gurung, R. Shah, and D. P. Jaiswal, "Software Development Life Cycle Models-A Comparative Study," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, no. March 2021, pp. 30–37, 2020, doi: 10.32628/cseit206410.

[22] S. M. Khan, "Waterfall Model Used in Software Development Reference: Software Requirements Engineering Waterfall Model," 2023. doi: 10.13140/RG.2.2.29580.69764.

[23] V. K. Sharma and A. K. S, "Hierarchical Cloud-IoT Architecture for AI-Powered Intelligent Disaster Response," in *2025 7th International Conference on Innovative Data Communication Technologies and Application (ICIDCA)*, IEEE, Oct. 2025, pp. 603–610. doi: 10.1109/ICIDCA66325.2025.11280408.

[24] A. Saravanos and M. X. Curinga, "Simulating the Software Development Lifecycle: The Waterfall Model," *Appl. Syst. Innov.*, vol. 6, no. 6, 2023, doi: 10.3390/asi6060108.

[25] H. P. Cyril, N. D. Bhandarwar, S. Kumara, and S. Mathur, "Securing Containerised Applications Using Kubernetes-Based Orchestration in Software Development," in *2026 International Conference on Intelligent Computing, Networks, and Security (IC-ICNS)*, Bhubaneswar, India: IEEE, 2026, pp. 1–6, March. doi: 10.1109/IC-ICNS68863.2026.11537887.

[26] G. Regulwar, P. Jawandhiya, V. Gulhane, and R. Tugnayat, "Variations in V Model for Software Development," 2021.

[27] I. M. Ibrahim, "Iterative and Incremental Development Analysis Study of Vocational Career Information Systems," *Int. J. Softw. Eng. Appl.*, vol. 11, no. 5, pp. 13–24, Sep. 2020, doi: 10.5121/ijsea.2020.11502.

[28] R. rao Thallada, N. Alapati, and K. Nallabothu, "Explainable AI-Driven Predictive Risk Management Framework for Enterprise GRC Platforms," 2026. doi: https://doi.org/10.21203/rs.3.rs-9328473/v1.

[29] P. Sankhe, N. Patil, N. Patwari, A. Agar, and R. Asati, "Evolution and Current Trends in Agile Software Development Methodologies: A Comprehensive Analysis of Industry Adoption and Practices," *IJARCCE*, vol. 14, 2025, doi: 10.17148/IJARCCE.2025.1411132.

[30] M. Ogunbukola, "Agile vs. Waterfall Methodologies in Public Sector Projects: A Comparative Analysis," 2024.

[31] A. Omonije, "Agile Methodology: A Comprehensive Impact on Modern Business Operations," *Int. J. Sci. Res.*, vol. 13, 2024, doi: 10.21275/SR24130104148.

[32] S. Kumar and R. Rishi, "Software Quality and Productivity Enhancement Model," vol. 12, no. 11, pp. 73–81, 2016.

[33] H. Cao, "Comparative Analysis of Software Development Methodologies," *Appl. Comput. Eng.*, vol. 190, pp. 21–27, 2025, doi: 10.54254/2755-2721/2026.TJ27796.

[34] M. Mumtaz, A. Khan, J. Koskula, J.-J. Luukkonen, and A. K. Mohammed, "Waterfall to DevOps transition Successful DevOps Driven Digital Transformation," 2024. doi: 10.13140/RG.2.2.21359.00169.

[35] P. Narang and P. Mittal, "Trends of Software Development Methodologies Toward DevOps: Analysis and Review," *Recent Adv. Comput. Sci. Commun.*, vol. 16, 2023, doi: 10.2174/2666255816666230619121018.

[36] J. A. Tsapa, "Integrating DevOps with Agile and other Software Development Methodologies," vol. 3, p. 5, 2022.

[37] S. Al Masud, M. Masnun, A. Sultana, A. Sultana, F. Ahmed, and N. Begum, "DevOps Enabled Agile: Combining Agile and DevOps Methodologies for Software Development," *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, 2022, doi: 10.14569/IJACSA.2022.0131131.

[38] M. Gokarna and R. Singh, "DevOps: A Historical Review and Future Works," in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, IEEE, Feb. 2021, pp. 366–371. doi: 10.1109/ICCCIS51004.2021.9397235.