iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80575

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

Smarter Software Engineering: Evaluating Classical vs Cognitive Lifecycles with Proposals

Abhinav Gupta[1], Dr. Sweety Maniar[2]
[1]*Phd Scholar, Department of IT & Computer Science, Swaminarayan University, Kalol, Gujarat. & India*
[2]*Phd Supervisor, Department of IT & Computer Science, Swaminarayan University, Kalol, Gujarat. & India*

Abstract: ***Conventional Software Development Lifecycle (SDLC) models offer a systematic framework for coordinating software engineering activities. Over time, numerous models have emerged, each characterized by distinct strengths and inherent limitations. This study aims to comparatively evaluate prominent SDLC paradigms, examine their fundamental characteristics, and propose a novel framework that incorporates computational intelligence principles to enhance efficiency, adaptability, and effectiveness. Through a comprehensive analysis of existing models and techniques such as machine learning and optimization algorithms, the study highlights strategies to overcome traditional SDLC challenges via improved decision-making, automated processes, and optimized resource allocation across the entire lifecycle.***
Keywords: Software Development Lifecycle (SDLC), Software Engineering,VModel, Agile Methodology, Waterfall Model, Spiral Model.

I. INTRODUCTION

Software development is an evolving discipline driven by continuous advancements in technologies and methodologies. The Software Development Lifecycle (SDLC) provides a structured framework for managing software projects, with models such as Waterfall, Agile, Spiral, and V-Model widely adopted. However, traditional approaches exhibit limitations that can be addressed through the integration of computational intelligence techniques, including artificial intelligence, machine learning, and optimization algorithms.

These techniques enhance critical aspects of software development, such as resource allocation, scheduling, risk management, and decision-making, improving overall process efficiency. This paper presents a comparative analysis of existing SDLC models to evaluate their strengths and limitations and proposes a novel SDLC framework incorporating computational intelligence to improve adaptability, scalability, and predictive performance.

The study covers both traditional SDLC models and computational intelligence methods such as neural networks, genetic algorithms, fuzzy logic, and metaheuristics. While promising, these approaches introduce challenges related to complexity and expertise requirements. The paper is structured to include a review of existing models, comparative evaluation, integration of computational intelligence, proposal of a new model, and concluding recommendations.

II. SDLC MODELS

Software Development Lifecycle (SDLC) models have long provided structured methods for managing software projects. These models outline sequential or iterative processes to guide development, with common examples including Waterfall, Iterative, Agile, Spiral, and V-Model.

1) Waterfall follows a rigid sequence of stages, offering simplicity but limited flexibility.
2) Iterative introduces adaptability by repeating cycles, though it can become complex in large projects.
3) Agile emphasizes collaboration, customer feedback, and rapid delivery, but requires cultural adjustments.
4) Spiral integrates risk analysis into iterative cycles, making it suitable for complex projects but costly.
5) V-Model enhances Waterfall with parallel testing phases, ensuring quality but still lacking flexibility.

Model	Core Approach	Key Strengths	Key Weaknesses	Best For
Waterfall	Fixed linear	Simple,	Inflexible;	Predictable,

Model	Core Approach	Key Strengths	Key Weaknesses	Best For
	sequence: requirements, design, development, testing, deployment, maintenance.	easy to manage and track.	struggles with mid-process changes.	small projects.
Iterative	Breaks project into cycles repeating Waterfall-like steps with feedback loops.	More adaptable than Waterfall.	Can get complex with too many iterations in large-scale efforts.	Projects needing incremental improvements.
Agile	Short cycles emphasizing team collaboration, customer input, and incremental delivery (e.g., Scrum, Kanban).	Highly flexible and responsive to change.	Requires organizational culture shift; less suited to rigid environments.	Dynamic, customer-driven projects.
Spiral	Cyclic blend of Waterfall/Iterative, prioritizing early risk identification and mitigation.	Excellent for handling uncertainties.	Higher cost and management complexity.	Large, high-risk, complex endeavors.
V-Model	Waterfall extension with mirrored testing phases per development stage (V-shaped).	Robust quality control via integrated testing.	Lacks flexibility for evolving requirements.	Projects demanding thorough verification.

III. COMPARATIVE ANALYSIS OF EXISTING SDLC MODELS:

Before undertaking a comparison of Software Development Lifecycle (SDLC) models, it is necessary to establish clear evaluation criteria. Typical factors include adaptability to evolving requirements, scalability, speed of development, risk management, stakeholder involvement, and overall project success. These benchmarks provide a structured basis for identifying the strengths and weaknesses of each model. The comparative analysis reveals that each SDLC model offers unique advantages and limitations. Waterfall provides clarity and structure but struggles with change. Agile emphasizes flexibility, collaboration, and customer feedback, though scalability and documentation may be challenging. Iterative enables refinement through repeated cycles while maintaining order. Spiral integrates risk management early but introduces complexity. V-Model ensures rigorous testing and quality assurance but lacks adaptability. Recognizing these trade-offs is essential for selecting the most appropriate model for specific project requirements.

Aspect	Criteria	Waterfall	Agile	Iterative	Spiral	V-Model
Key Traits	- Adaptability - Scalability - Speed - Risks - Collaboration - Success	Structured, simple	Flexible, collaborative	Cycle improvements	Risk-focused cycles	Testing emphasis
Strengths	Easy to manage	Teamwork & feedback	Balanced structure	Early risk mitigation	Quality assurance	
Limitation	Poor with changes	Scaling/docs issues	-	Complexity	Low flexibility	

IV. TOWARD A NEW SDLC PARADIGM – THE PROPOSAL

The limitations of conventional Software Development Lifecycle (SDLC) models—particularly their inability to effectively manage growing complexity, rapid technological evolution, and shifting user requirements—necessitate the development of a new paradigm. This proposed framework integrates the structured discipline of traditional approaches with the adaptability of modern methodologies, further strengthened by computational intelligence techniques such as artificial intelligence, machine learning, and optimization. The model supports predictive decision-making, automated task execution, and optimized resource allocation. Unlike rigid predecessors, it emphasizes continuous learning, real-time feedback, and adaptive planning, thereby improving software quality, reducing development time and cost, and offering scalable resilience for future software engineering

V. APPLYING THE MODEL: PRACTICAL INSIGHTS

The application of the proposed SDLC model in real-world projects demonstrates its effectiveness and adaptability. By leveraging data-driven techniques, the model enables improved decision-making through accurate risk prediction, efficient resource management, and streamlined development processes. Its iterative and intelligent framework supports continuous feedback, allowing for the refinement of requirements and enhancement of product quality throughout the lifecycle. Furthermore, the incorporation of automation minimizes manual effort in areas such as testing, deployment, and monitoring, thereby accelerating delivery and improving system reliability. However, successful implementation necessitates proper planning, robust infrastructure, and skilled professionals with expertise in advanced computational methods. Overall, this approach enhances development efficiency, scalability, and performance.

VI. PERFORMANCE ASSESSMENT AND VALIDATION

For the successful adoption of the proposed SDLC model, rigorous evaluation and validation are essential. Evaluation focuses on assessing whether the model achieves key objectives such as improved development speed, enhanced product quality, effective resource utilization, and stakeholder satisfaction. Continuous monitoring through checkpoints and feedback mechanisms ensures alignment with project goals and facilitates ongoing improvement. Validation, on the other hand, involves practical testing through pilot implementations, simulations, or proof-of-concept studies to examine the model's real-world performance. Proper documentation of outcomes promotes transparency and organizational learning. Moreover, validation requires collaborative involvement from developers, testers, project managers, analysts, and end-users. This collective effort strengthens confidence in the model, identifies potential limitations, and supports continuous refinement, ensuring its robustness in modern software development environments.

VII. WATERFALL AND AGILE MODEL PROJECT COMPARISION RELATED TO COST AND TIME

1) Requirements: 2 months → ₹20 lakh
2) Design: 2 months → ₹30 lakh
3) Development: 3 months → ₹30 lakh

4) Testing: 1 month → ₹20 lakh
5) Deployment: End of month 8

A. Waterfall Model (Sequential)

Planned Timeline & Cost

- Total Duration = 8 months
- Total Budget = ₹100 lakh

Formulae

1. Cost of Change (CoC)
   CoC=Base Cost×Change Factor
   Suppose a requirement change occurs in month 6 (after design & development).
   - Base Cost = ₹100 lakh
   - Change Factor ≈ 0.25 (late changes are expensive)

   CoC=100,00,000×0.25=₹25,00,000

2. Delay Calculation
   Delay=Rework EffortTeam Productivity
   - Rework Effort = 2 months
   - Team Productivity = 1 month per sprint equivalent

   Delay=2 months

Result:

- Final Cost = ₹125 lakh
- Final Duration = 10 months

B. Agile Model (Iterative)

Sprint Allocation

- 8 sprints of 1 month each
- Deliverables after each sprint (incremental features)

Formulae

1. Cost of Change (CoC):
   CoC=Base Cost×Change Factor
   Suppose the same requirement change occurs in Sprint 6.
   - Base Cost = ₹100 lakh
   - Change Factor ≈ 0.10 (easier to adapt mid-sprint)

   CoC=100,00,000×0.10=₹10,00,000

2. Delay Calculation:
   Delay=Rework EffortTeam Productivity
   - Rework Effort = 0.5 month
   - Team Productivity = 1 month per sprint equivalent

   Delay=0.5 months

Result:

- Final Cost = ₹110 lakh
- Final Duration = 8.5 months

VIII. RESULTS COMPARISION

Metric	Waterfall	Agile
Planned Duration	8 months	8 months
Planned Budget	₹100 lakh	₹100 lakh
Change Request Cost	₹25 lakh (25%)	₹10 lakh (10%)
Delay	2 months	0.5 month
Final Cost	₹125 lakh	₹110 lakh
Final Duration	10 months	8.5 months

IX. CONCLUSION

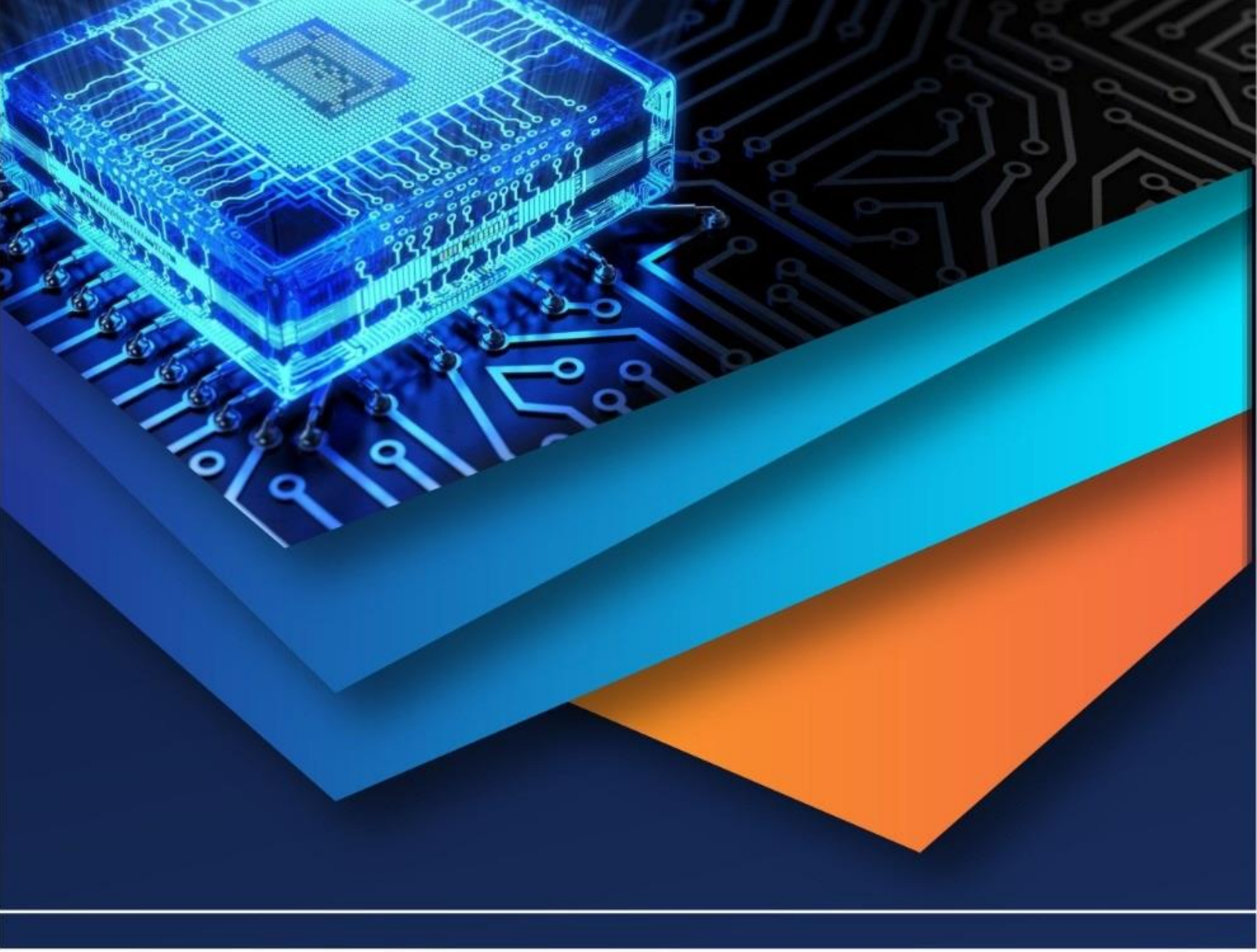
The limitations of conventional SDLC models become increasingly evident in the context of dynamic, high-velocity software ecosystems. While models such as Waterfall, Iterative, Agile, Spiral, and V-Model provide foundational methodologies characterized by structure, feedback integration, adaptability, risk management, and validation rigor, they exhibit constraints when addressing large-scale uncertainty and rapid change.

Cognitive SDLC paradigms, driven by artificial intelligence, machine learning, and optimization frameworks, introduce a transformative layer of intelligence into development processes. These systems enable predictive analytics, autonomous decision support, and continuous optimization of workflows. The proposed hybrid model operationalizes this integration, enhancing resource allocation strategies, enabling proactive risk simulation, and supporting iterative refinement at scale. However, the transition necessitates addressing computational overhead, skill gaps, and infrastructure scalability.

In synthesis, the convergence of classical engineering discipline and cognitive intelligence defines the next generation of software development, enabling resilient, scalable, and high-performance systems capable of thriving in complex technological landscapes.

REFERENCES

[1] Yas, Q. M., ALazzawi, A., & Rahmatullah, B. (2023). A Comprehensive Review of Software Development Life Cycle methodologies: Pros, Cons, and Future Directions.
[2] Cinkusz, K., et al. (2025, Aug 21). Cognitive Agents Powered by Large Language Models for Agile Software Project Management
[3] Swann, C. (2024-2025). AI-Native Software Development Lifecycle (SDLC)
[4] Cognitive Software Engineering: A Research Framework by M. J. van der Meulen (2014)
[5] Comparison of Different Life Cycle Models in Software Engineering (2023)
[6] A Comparative Study of Software Development Models: Evolution, Strengths, and Modern Applications (2024)



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY