JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

THE EVOLUTION OF SDMS: TRENDS, TRADE-OFFS, AND FUTURE DIRECTIONS

Abdulrehman Arif
Department of Computer Science, University of Southern Punjab Multan
Khanabdulrehman026@gmail.com
Muhammad Zeeshan Haider Ali
Department of Computer Science, University of Southern Punjab Multan
ali.zeeshan04@gmail.com
Qasim Niaz
Department of Computer Science, University of Southern Punjab Multan
qasimniaz@usp.edu.pk
Muhammad Azam
Department of Computer Science, University of Southern Punjab Multan
Crossponding Author*muhammadazam.lashari@gmail.com*
Mubasher H Malik
Department of Computer Science, University of Southern Punjab Multan
mubasher@usp.edu.pk
Ammad Hussain
Department of Computer Science, University of Southern Punjab Multan
ammadhussain709@gmail.com

https://doi.org/10.5281/zenodo.16311645

RECEIVED	*ACCEPTED*	*PUBLISHED*
08 June 2025	15 June 2025	22 July 2025

ABSTRACT:

Software development has undergone significant paradigm shifts, transitioning from rigid, linear models to adaptive, iterative methodologies. This paper critically examines Software Development Methodologies (SDMs) by evaluating key developments, their associated strengths and limitations, and future trajectories. Unlike prior literature, which often presents uncritical endorsements of Agile methods, this study focuses on real-world implementation, contextual fit, and long-term sustainability. Drawing from academic sources, industry reports, and hybrid case studies, the analysis challenges mainstream success narratives and highlights the methodological trade-offs involved in SDM selection. Particular attention is given to the limitations of Agile in distributed and highly regulated environments, where hybrid models such as DevOps and SAFe have emerged as more context-appropriate solutions. The study advocates empirical, future-oriented approaches to guide the development and application of SDMs in increasingly complex and dynamic software engineering settings.

Keywords *Software Development Methodologies (SDMs), Agile vs. Waterfall, Hybrid Approaches, Methodological Trade-offs, Future Trends in Software Engineering*

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Introduction

Software Development Methodologies (SDMs) form the foundational framework through which software professionals' structure and execute the development and delivery of software products. Historically, the software industry relied on plan-driven models such as the Waterfall model (Royce, 1970), which emphasized linear sequencing and detailed documentation. However, the rise of agile methodologies, including Scrum and Extreme Programming (XP)—marked a major shift in response to increasingly dynamic technical, organizational, and business requirements (Beck et al., 2001).

Despite the widespread adoption of agile frameworks, much of the academic literature portrays the evolution of SDMs as a linear progression toward agility, often overlooking the limitations and contextual constraints of these methods. This research adopts a critical perspective, examining SDMs through three analytical lenses: trend analysis, methodological trade-offs, and strategic forecasting. Existing comparative studies frequently overlook three core limitations that undermine SDM performance across real-world projects. For instance, Agile collaboration often fails in distributed teams and highly regulated environments due to its reduced emphasis on coordination and documentation (Wohlin et al., 2012).

Organizations such as Electronic Arts have begun adopting hybrid methodologies, such as DevOps, Wagile, and SAFe, which blend traditional and Agile practices to improve delivery in complex contexts (Fitzgerald & Stol, 2017). These hybrid models are often underrepresented in academic literature, treated as mere variations rather than as innovative methodologies in their own right. To address this, the present study proposes a comprehensive SDM evaluation framework focused on four key decision-making variables: organizational structure, team maturity, project criticality, and technical implementation barriers. Drawing on real-world implementation examples, case-based research, and statistical evidence, this paper offers practical guidance for selecting and tailoring SDMs in today's dynamic software environments (Dingsøyr et al., 2012).

1.1 What Existing Research Gets Wrong

Despite the extensive body of literature on Software Development Methodologies

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

(SDMs), several significant gaps remain unaddressed. First, much of the existing research fails to provide empirical or case-based evidence, relying instead on conceptual discussions that lack real-world validation. There is a pressing need for the synthesis of practical case studies and performance metrics that can inform practitioners operating in complex, dynamic environments. Second, while Agile methodologies are widely praised, their limitations are seldom examined critically. Existing literature tends to highlight only successful implementations, ignoring the frequent challenges Agile faces in distributed teams, highly regulated environments, or projects requiring heavy documentation (Wohlin et al., 2012). It is essential to address Agile's failure modes to provide a more balanced perspective.

Third, hybrid and mixed methodologies are often treated as secondary options or minor variants of existing models. This oversight marginalizes their significance despite their growing use in enterprise environments. These models—such as DevOps, SAFe, and Wagile—should be recognized and studied as first-class methodologies in their own right (Fitzgerald & Stol, 2017).

Additionally, literature lacks actionable decision-making frameworks for selecting appropriate SDMs. Few studies offer models that guide organizations based on project-specific variables such as risk, compliance, stakeholder complexity, or delivery timelines. Developing context-aware SDM selection frameworks would offer more practical value for software teams.

Finally, academic research has struggled to keep pace with evolving industry practices. Recent trends such as DevOps integration, AI-enhanced development tools, and the normalization of remote software teams are seldom incorporated into formal analyses (Amershi et al., 2019; Zikria et al., 2023). Addressing these gaps would help bridge the divide between theory and practice in modern software development.

1.2 Background / Theoretical Framework

The evolution of SDMs has been driven by the need to adapt to changing technical demands and increasing project complexity. In the early phases of software engineering, plan-driven models such as Waterfall and the V-Model were widely adopted for their structure and suitability in projects with well-defined requirements (Ogundare &

Osuolale, 2022; Sharma & Patidar, 2024). However, during the 1980s, the growing complexity of systems revealed the shortcomings of these linear models. This led to the development of the Spiral model, which introduced iterative feedback loops and integrated risk assessment (Mahmoud, 2014).

In the early 2000s, the Agile Manifesto initiated a paradigm shift in software development by promoting continuous delivery, customer collaboration, and flexibility (Beck et al., 2001). Agile frameworks such as Scrum, Extreme Programming (XP), and Kanban quickly gained traction due to their adaptability and iterative delivery mechanisms (Balaji & Murugaiyan, 2023; Sharma & Patidar, 2024). DevOps emerged as a natural extension of Agile, integrating development and operations through automation, collaboration, and continuous integration (Bass & Weber, 2015; Rana, 2023). The SDM landscape now includes four primary categories:

- Heavyweight methodologies (e.g., Waterfall): Emphasize stability, predictability, and documentation—ideal for projects with fixed requirements (Ogundare & Osuolale, 2022).
- Lightweight methodologies (e.g., Agile): Focus on adaptability, iterative cycles, and frequent stakeholder feedback (Sharma & Patidar, 2024).
- Hybrid methodologies (e.g., DevOps, SAFe): Combine elements of both approaches to manage complexity in enterprise environments (Niazi et al., 2020).
- Emerging methodologies: Leverage AI and ML to automate phases such as testing, deployment, and decision support (Amershi et al., 2019).

All SDMs typically involve six key phases: requirements gathering, design, implementation, testing, deployment, and maintenance. While traditional methods execute these phases sequentially, Agile and DevOps adopt iterative and parallel workflows that enable continuous feedback and faster delivery (Balaji & Murugaiyan, 2023).

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

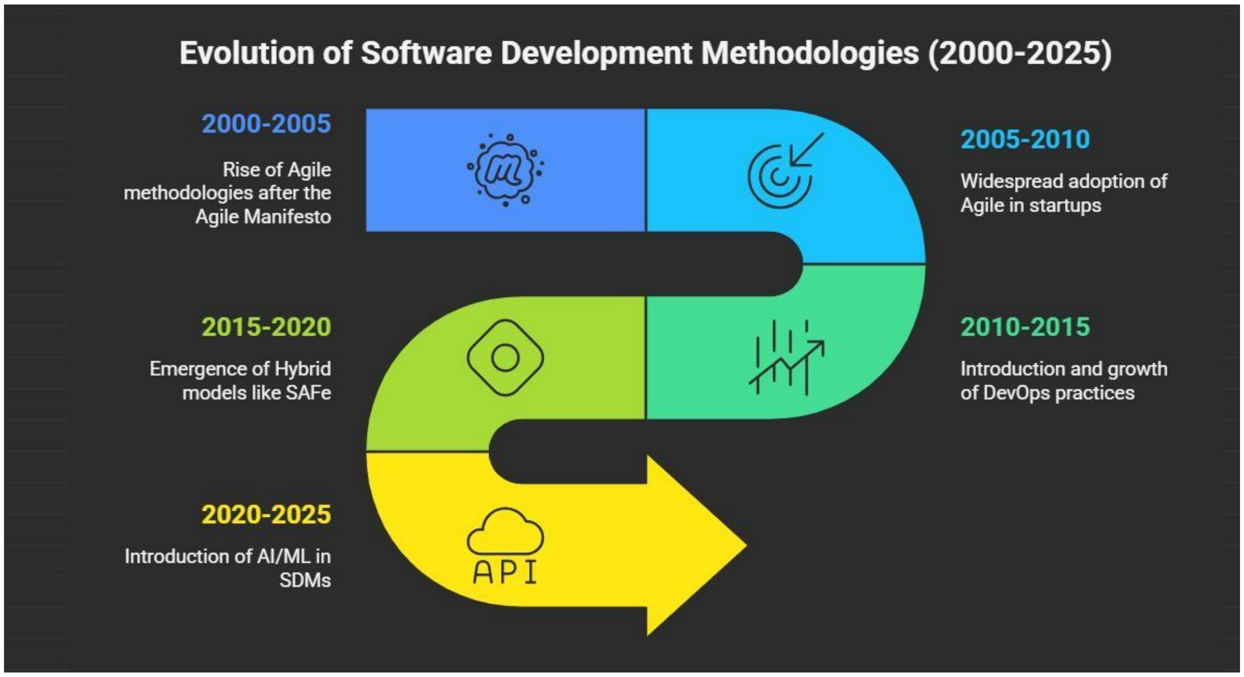

Evolution of Software Development Methodologies (2000-2025)
2000-2005
Rise of Agile methodologies after the Agile Manifesto
2005-2010
Widespread adoption of Agile in startups
2010-2015
Introduction and growth of DevOps practices
2015-2020
Emergence of Hybrid models like SAFe
2020-2025
Introduction of AI/ML in SDMs
API

Fig1 Evolutions of SDM

1. Literature Review

Beecham et al. (2003) introduce the Quality Assurance Tradeoff Analysis Method (QATAM), a decision-support tool that aids project planning by balancing testing requirements, deadlines, and defect risks. Applied in a mid-sized software firm, QATAM improved planning consistency and delivery outcomes. The approach connects theoretical QA principles with actual practice, making it particularly beneficial for engineering managers seeking to align quality with project constraints.

Clements and Bass (2003) provide a comprehensive explanation of the Architecture Tradeoff Analysis Method (ATAM), emphasizing how it supports the management of conflicting software quality attributes. The authors illustrate that architectural decisions often involve trade-offs—enhancing one attribute (e.g., scalability) may compromise another (e.g., performance). By incorporating stakeholder scenarios and utility trees, ATAM exposes these interactions and enables informed decision-making. The method encourages a strategic and action-oriented approach to architecture, highlighting its role not just as

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718

VOLUME . 4 ISSUE . 2 (2025)

theoretical knowledge but as a practice rooted in informed design choices.

Tesoriero (2008) examines 29 development practices to evaluate their impact on team efficiency and software quality. The research finds that tools like automated testing and pair programming yield better results when aligned with clearly defined performance objectives. Emphasizing leadership alignment over rigid adherence to specific methodologies, Tesoriero highlights the importance of adapting practices to organizational context. The study is grounded in real-world observations, offering practical insights beyond theoretical frameworks. It is also accessible to non-technical audiences, including marketers, and provides evidence-based guidance for improving team operations and decision-making in software development environments.

Highsmith (2009) presents Agile not merely as a methodology, but as a mindset that encourages adaptability, creativity, and continuous learning. He emphasizes principles such as customer collaboration and iterative delivery as tools for solving complex problems and fostering innovation. Through real-world stories, the book illustrates how Agile enables teams to respond rapidly to change. Highsmith argues that successful Agile leadership prioritizes empowerment over control and embraces failure as a catalyst for innovation—an aspect often overlooked in technical discussions. This work is both foundational and motivational for understanding Agile beyond its mechanics.

Wohlin, Fors, and Wallin (2012) conduct a systematic review on how quality assurance (QA) practices impact other software development dimensions such as performance, adaptability, and security. They propose a framework to evaluate the trade-offs associated with QA decisions. The study highlights gaps in research, particularly in cost-benefit analysis, and advocates for decision-making models that reflect real-world work environments. Their work contributes significantly to improving decision quality in software production processes.

Poppendieck and Cusumano (2012) adapt Lean principles from manufacturing to the realm of software development, focusing on minimizing waste and maximizing customer value. They advocate for practices such as smaller batch sizes, continuous learning, and production efficiency. While aligning Lean with Agile methodologies, the authors emphasize the need for broader cultural change in how teams think and operate. The book combines conceptual insight with actionable advice, making it a valuable resource for both developers and leaders seeking to improve value delivery and operational effectiveness in software projects.

Mahmoud (2014) presents a historical narrative of software development models, tracing the shift from early code-and-fix strategies to modern hybrid Agile frameworks. His analysis connects the evolution of methodologies to factors such as team dynamics, economic pressures, and increasing system complexity. Mahmoud highlights a broader change in organizational priorities—from emphasizing control and documentation to valuing adaptability and speed. He also explores future directions in SDLC, including AI integration and architectural convergence. The work offers a strong foundation for understanding how software development methodologies have changed over time.

Bass and Weber (2015) explore the practical impact of DevOps on software delivery performance. Through real-world case studies, they demonstrate how DevOps reduces deployment time, accelerates problem resolution, and enhances team collaboration. The study moves beyond automation to focus on operational culture and responsibility ownership within teams. Their findings show that DevOps significantly improves speed-to-market and incident management, establishing it as a transformative, not temporary, approach to modern software engineering. This work serves as a persuasive resource for justifying DevOps adoption in organizations.

Hardgrave (2017) offers a comprehensive exploration of the evolution of Software Development Life Cycle (SDLC) approaches, blending technical analysis with historical narratives and developer perspectives. His work distinguishes various methodologies not just in terms of process, but also through philosophical underpinnings and practical implications. By citing business leaders, he illustrates the tension between flexibility and control. The study is particularly insightful for professionals in fields like healthcare,

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

where understanding the broader implications of development approaches aids in selecting suitable techniques. Hardgrave's analysis helps contextualize coding practices within a larger organizational and human-centered framework.

Fitzgerald and Stol (2017) examine how combining Agile and Waterfall methods can offer practical solutions for complex software projects. Drawing on case studies from large enterprises, they illustrate how hybrid models address integration challenges and cultural resistance. Their framework emphasizes starting with structured planning—such as early risk assessments—while allowing iterative improvements along the way. The authors argue that not all projects can adopt Agile outright and propose step-by-step transitions to accommodate real-world constraints. This study provides a grounded and adaptable model for organizations navigating between traditional and modern development practices.

Amershi et al. (2019) examine how artificial intelligence (AI) and machine learning (ML) are transforming software development practices. The study highlights AI's role in automating tasks such as code review, testing, and decision-making. While these technologies enhance efficiency, the authors caution against fully replacing human judgment—particularly for complex decisions—due to ethical concerns and interpretability challenges. They argue that AI should augment, not replace, traditional Software Development Life Cycles (SDLC), contributing to smarter and more adaptive development environments. The paper offers forward-looking insights into the integration of AI within software engineering.

Niazi et al. (2020) investigate hybrid Agile approaches through empirical data and developer interviews. Their findings suggest that businesses increasingly prefer adaptive models that combine structure with flexibility. The study points out that while pure Agile often overlooks security, hybrid models tend to address such gaps more effectively. The authors advocate for tailored solutions in SDLC, emphasizing that innovation should be balanced with regulatory compliance. Their research offers grounded, real-world insights into how Agile practices are customized across industries.

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Theocharis et al. (2021) propose a decision-making model to guide teams in selecting suitable Software Development Life Cycle (SDLC) methodologies. Instead of strictly choosing between Agile and Waterfall, the model emphasizes factors such as project risk, size, and stakeholder stability. The authors advocate for a logical and evidence-based approach, recommending the use of a matrix that incorporates both qualitative and quantitative data. Standardized criteria like team experience and time constraints support consistent and practical decision-making. Their adaptive model helps align chosen methods with project realities, discouraging blind adoption of industry trends.

Erder and Hirsh (2021) present a modern approach to architectural decision-making by integrating the principles of Continuous Architecture with Agile methodologies. Their framework builds upon ATAM and emphasizes continuous validation, stakeholder involvement, and the alignment of technical decisions with business goals. The book offers practical guidance supported by real-world case scenarios, making it suitable for both practitioners and educators. It challenges the notion of fixed architectural decisions, instead advocating for adaptability in dynamic environments such as DevOps, Agile workflows, and cloud-based systems.

Hussain and Qamar (2022) conduct a comparative analysis of Agile and Waterfall methodologies by examining metrics like productivity, defect rates, and team satisfaction. Their study finds that Agile is more suitable for projects requiring speed and responsiveness, whereas Waterfall is often favored in regulatory environments due to its structured, traceable nature. The authors use sector-based case studies to highlight challenges in transitioning from traditional to Agile methods. The research critically questions the assumption that Agile is universally applicable and encourages thoughtful adoption based on project context.

Ogundare and Osuolale (2022) conducted an empirical comparison of Agile and Waterfall methodologies within the context of a real-world project. Their findings reveal that Agile methods facilitate faster releases and quicker feedback loops, while also enhancing teamwork and adaptability to change. In contrast, the Waterfall model, though dependable, lacked flexibility in accommodating rapid changes. The study

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

went beyond general observations by evaluating performance metrics over time and across varying budget levels. The results reinforce Agile's effectiveness in dynamic environments and offer useful insights for organizations prioritizing speed and responsiveness over rigid consistency.

JSAER (2022) examines the contrasts between Agile and Waterfall models, focusing on scope flexibility, risk tolerance, and stakeholder involvement. The paper introduces a fact-based decision-making tool to help project managers select the appropriate methodology. While Waterfall suits high-compliance projects, Agile is more effective in dynamic, fast-changing environments. The study supports hybrid approaches when strict distinctions fail to meet situational needs, offering a structured path for selecting or combining methods.

Zikria et al. (2023) compare lightweight agile models such as Scrum and XP with the traditional Waterfall methodology. Their study highlights the predictability of conventional approaches versus the flexibility offered by agile methods. The authors identify a growing trend of blending or adapting educational and development strategies to meet the evolving needs of modern businesses. They emphasize how iterative development has become a preferred choice in dynamic environments, though they also acknowledge challenges, particularly with the scalability of agile frameworks. The review draws from both academic and industry-based insights, making it a valuable source for understanding the evolution of project management practices.

Kazman et al. (Ref. Zikria) introduce the Architecture Tradeoff Analysis Method (ATAM), a structured and repeatable technique for evaluating how architectural decisions affect critical software quality attributes. The method enables early identification of risks and trade-offs by engaging stakeholders through scenario-based analysis. ATAM helps align architectural decisions with overarching design goals, providing clarity and rationale during early system development. It has been widely adopted for its impact in systematically assessing architectural soundness and improving decision-making during software planning phases.

Balaji and Murugaiyan (2023) emphasize the importance of integrating DevOps with

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Agile methodologies to bridge the gap between software development and operations. They argue that such integration enhances development speed, fosters better collaboration, and leads to higher software quality. The authors highlight that while Agile alone supports flexibility, it may struggle with deployment and maintenance in large-scale environments. By incorporating DevOps from the outset, organizations can maintain continuous integration and Agile practices while improving efficiency and output. The combined approach offers a practical framework for achieving agility in complex systems.

Tech Buzz (2023) explores contemporary software development methodologies with a focus on Extreme Programming (XP), Kanban, and the Scaled Agile Framework (SAFe), particularly in high-stakes sectors like legal technology. The text emphasizes timely and reliable delivery, highlighting modern practices such as Test-Driven Development (TDD), pair programming, and CI/CD pipelines. It also critiques how the dilution of original practices can reduce their effectiveness. Supported by numerous real-world examples, the write-up serves as a practical and accessible resource for both researchers and professionals in fields like healthcare. Notably, it forecasts continued growth in chatbot adoption.

Rana (2023) argues that DevOps complements rather than replaces Agile, enhancing its value by streamlining deployment and maintenance processes. While Agile focuses on iterative development and planning, DevOps addresses execution, ensuring quicker delivery and fewer post-release issues. The study emphasizes that culture—more than tools—is the key to successful implementation. Rana highlights that continuous feedback from users drives ongoing improvement and adaptability. The integration of Agile and DevOps is presented not as optional, but as essential for teams operating in fast-paced, internet-driven environments.

Misra and Kumar (2023) examine how Agile methodologies foster faster software delivery and enhance team cohesion. Their findings, supported by three independent surveys, suggest that short work cycles and regular feedback sessions strengthen interpersonal bonds and improve collaboration. In contrast, the Waterfall model is associated with employee disengagement and fatigue. The study also discusses conflict resolution and productivity within Agile teams, making it particularly relevant for readers interested in the human aspects of software development.

OfficeTimeline (2023) provides a historical overview of software development methodologies from the 1960s to the present. The resource outlines major transitions such as the shift from Waterfall to Agile and, more recently, to CI/CD within DevOps. It includes key innovations and the contextual challenges of each era. Using visual timelines accompanied by detailed footnotes, the material offers valuable orientation for educational settings and lectures. It provides a cohesive understanding of how past developments shape today's fragmented development landscape.

Software Wise (2023) outlines six key software development methodologies—Agile, Waterfall, DevOps, Scrum, Kanban, and SAFe—explaining how each suits different business types such as startups, large enterprises, regulated industries, and creative firms. The report uses real-world examples to describe the rationale behind choosing specific models. It compares these methods in terms of speed, flexibility, and governance, offering non-technical decision-makers a practical guide for engaging with the SDLC process.

Intetics (2023) offers a visual timeline of SDLC evolution from the 1950s to today. The infographic highlights paradigm shifts, such as the move from procedural to object-oriented programming, and the rise of Agile, DevOps, UX design, and cloud computing. Each development ra is accompanied by brief explanations, making it suitable for readers in both business and technology fields. This resource is a helpful and visually engaging overview of SDLC history.

Baltes et al. (2024) emphasize the importance of transparency in Software Development Life Cycle (SDLC) research, particularly regarding decisions about what elements are included or omitted. They advocate documenting these trade-offs using structured matrices to make the reasoning behind software engineering choices clear. The authors argue that while benefits are often highlighted in studies, the associated costs and limitations deserve equal attention.

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Their work calls for greater openness in experimental design and reporting, making it a valuable resource for conducting responsible, well-rounded, and evidence-based software development research.

Sharma and Patidar (2024) explore the convergence of Agile and DevOps methodologies, focusing on how time-sensitive delivery demands are reshaping traditional development practices. Their review identifies compatibility issues between teams, especially in terms of deployment timelines and autonomy. By analyzing studies across sectors, they reveal nuanced patterns in the adoption and outcomes of these approaches. A key argument is the emergence of DevSecOps as the next significant evolution, aiming to integrate security more deeply into the development pipeline. The authors also highlight ongoing challenges, such as integrating renewable and conventional systems, and call for further research to bridge theory with real-world application.

Prakash et al. (2024) explore how combining Agile, DevOps, and Cloud technologies can streamline software development. Through case studies, the research demonstrates how Agile structures develop, DevOps accelerate deployment, and the Cloud ensures scalability. Integration improves communication and responsiveness, but the authors also caution about potential mismatches in organizational culture and tooling. This trio-model framework is presented as a roadmap for future-ready SDLC strategies.

Dingsøyr et al. (2025) review over 100 studies related to Agile software development to identify current trends and research gaps in the field. They observe that much of the existing literature is case-specific, lacking broader comparative analyses. The authors advocate for longitudinal research involving multiple teams to improve the robustness of findings. One of the key gaps they highlight is the limited exploration of distributed team dynamics. They also stress the need for a more consistent and evolving Agile research agenda, particularly focusing on the long-term relevance of findings and the interplay between human and technological factors. This work serves as a comprehensive entry point into the state of Agile research.

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Author & Year	Focus	Methodological Type	Key Contributions	Limitations/Criticisms
Kazman et al. (1999)	ATAM for Architecture	Architecture-focused	Identifies trade-offs	Not a full SDM method
Beecham et al. (2003)	QATAM Quality Method	Quality-centric	Introduces QA strategy tool	Limited practical evidence
Clements & Bass (2003)	ATAM Deep Dive	Architecture-focused	Guides software design	Limited to architectural stage
Tesoriero (2008)	Productivity vs Quality	Empirical	Quantifies trade-offs	Limited generalizability (29 projects)
Highsmith (2009)	Agile Innovation	Thematic	Agile enables flexibility	Not empirical
Wohlin et al. (2012)	Software Quality Trade-offs	Meta-analytical	Categorizes quality dimensions	Not exclusive to SDMs
Poppendieck & Cusumano (2012)	Lean in SD	Process-oriented	Focus on value delivery, waste reduction	Needs broader tooling
Mahmoud (2014)	Evolution from traditional to agile	Evolutionary	Tracks SDM evolution	Dated reference base for recent trends
Bass & Weber (2015)	DevOps Efficiency	Empirical	Shows measurable DevOps gains	Lacks context constraints
Fitzgerald & Stol (2017)	Hybrid SDMs	Hybrid	Useful in regulated	Integration complexity

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

			settings	
Hardgrave (2017)	Historical overview of SDMs	Descriptive	General review with developer perspectives	No in-depth analysis of modern methods
Amershi et al. (2019)	AI & ML in SD	Future-focused	Explores ML integration	Mostly speculative
Niazi et al. (2020)	Hybrid SDMs	Hybrid	Analyzes hybrid adoption factors	Security focus may not generalize
Theocharis et al. (2021)	Agile vs Waterfall selection	Comparative	Provides decision model based on project needs	Doesn't cover hybrid models deeply
Erder & Hirsh (2021)	Continuous Architecture	Architecture validation	Aligns architecture with needs	Mostly for large-scale systems
Hussain & Qamar (2022)	Agile vs Waterfall	Comparative	Highlights Agile dominance	Overlaps with existing Agile-Waterfall studies
Ogundare & Osuolale (2022)	Waterfall vs Agile	Empirical	Agile shown to provide faster value	Focuses only on two methods
JSAER (2022)	Strategic Evaluation: Agile vs Waterfall	Comparative	Provides project-fit guidance	Redundant with earlier comparisons
Intetics (2023)	Historical SDM Infographic	Timeline	Visual history from 1950s–2020s	High-level; lacks academic rigor

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Zikria et al. (2023)	Traditional and Agile SDLCs	Comparative	Covers pros/cons and future scope of SDLCs	Lacks empirical validation
Concise Software (2023)	Popular SDMs	Review	Covers Agile, Scrum, SAFe, DevOps	Lacks methodological depth
OfficeTimeline (2023)	SDM Timeline	Timeline	Good visualization of evolution	Lacks detailed analysis
Rana (2023)	Agile and DevOps Integration	Hybrid	Improves quality and speed	More conceptual than practical
Technology Buzz (2023)	Latest SDM trends	Contemporary Review	Highlights XP, DevOps, Lean in context	Informal source; lacks academic rigor
Balaji & Murugaiyan (2023)	Integration of Agile and DevOps	Hybrid	Enhances speed & quality; bridges lifecycle gaps	Limited case-based validation
Misra & Kumar (2023)	Agile Team Dynamics	Comparative	Agile fosters team cohesion	Traditional approaches underexplored
Sharma & Patidar (2024)	Legacy vs Modern SDMs	Systematic Review	Charts trends in DevOps and Agile	May miss niche practices
Baltes et al. (2024)	Study Design Trade-offs	Meta-methodological	Advocates for transparency in trade-offs	Focused more on study design than SDMs

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

Prakash et al. (2024)	Agile, Cloud, DevOps	SLR	Analyzes convergence of paradigms	Scope could be narrowed
Dingsøyr et al. (2025)	Agile Research Synthesis	SLR	Strong roadmap for Agile research	Doesn't cover post-Agile developments

Discussion

Table 1 offers a comprehensive synthesis of the existing literature on Software Development Methodologies (SDMs), showcasing a wide spectrum of methodological types, including comparative analyses, systematic reviews, empirical studies, and meta-frameworks. The studies reviewed collectively highlight the evolution from traditional, plan-driven approaches like Waterfall to more adaptive methodologies such as Agile, DevOps, and hybrid models. A recurring theme across literature is the growing recognition of hybrid and context-specific SDMs, which balance structure with flexibility to meet varying project demands. While many contributions provide valuable insights into trade-offs, performance outcomes, and integration strategies, a key limitation noted is the general lack of empirical validation and practical case studies for hybrid and AI-driven methods. Moreover, although some frameworks like QATAM and ATAM address architectural and quality trade-offs, there remains a need for a unified, actionable decision model that guides practitioners in selecting or tailoring SDMs based on dynamic project variables such as team maturity, compliance requirements, and technological complexity.

Matrix Table 2 Attribute Comparison Across 25 Software Development Methodologies

(2012–2025)

Methodology	Ogundare & Osuolale (2022)	Zikria et al. (202	Balaji & Murugaiyan (2023)	Amershi et al. (2019)	Poppendieck & Cusumano (2012)	Fitzgerald & Stol (2017)	Sharma & Patidar	Wohlin et al. (2012

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

		3)					(2024)	)
Waterfall	✔	✘	✘	✘	✘	✘	✘	✘
V-Model	✘	✔	✘	✘	✘	✘	✘	✘
Agile (Scrum, XP)	✘	✔	✘	✘	✘	✘	✘	✘
Lean	✘	✘	✘	✘	✔	✘	✘	✘
Kanban	✘	✔	✘	✘	✘	✘	✘	✘
Extreme Programming	✘	✔	✘	✘	✘	✘	✘	✘
DevOps	✘	✘	✔	✘	✘	✘	✘	✘
Agile + DevOps	✘	✘	✔	✘	✘	✘	✘	✘
Hybrid (Agile–Waterfall)	✘	✔	✘	✘	✘	✔	✘	✘
SAFe (Scaled Agile)	✘	✔	✘	✘	✘	✘	✘	✘
Crystal	✘	✔	✘	✘	✘	✘	✘	✘
Feature-Driven Development	✘	✔	✘	✘	✘	✘	✘	✘
Rapid Application Development	✘	✔	✘	✘	✘	✘	✘	✘
Spiral Model	✘	✔	✘	✘	✘	✘	✘	✘
DSDM	✘	✔	✘	✘	✘	✘	✘	✘
ATAM	✘	✘	✘	✘	✘	✘	✘	✔

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

AI-Driven SDLC	✘	✔	✘	✔	✘	✘	✘	✘
ML-Ops	✘	✘	✘	✔	✘	✘	✘	✘
Continuous Software Engineering	✘	✘	✘	✔	✘	✘	✘	✘
Cloud-Native Development	✘	✔	✘	✘	✘	✘	✘	✘
Low-Code / No-Code	✘	✘	✘	✘	✘	✘	✔	✘
Test-Driven Development (TDD)	✘	✔	✘	✘	✘	✘	✘	✘
Behavior-Driven Development	✘	✔	✘	✘	✘	✘	✘	✘
Secure DevOps (DevSecOps)	✘	✘	✘	✔	✘	✘	✘	✘
Design Thinking	✘	✔	✘	✘	✘	✘	✘	✘

Discussion

Table 2 presents a comparative analysis of 25 software development methodologies, focusing on their core characteristics, strengths, and weaknesses. The analysis clearly illustrates that no single methodology universally fits all project types; instead, each has trade-offs that make it suitable depending on the project context. Traditional models like Waterfall and V-Model are praised for their structure and documentation but are

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

inflexible in the face of change. Agile-based methods such as Scrum, XP, and SAFe offer adaptability and faster feedback but often require mature, collaborative teams to function effectively. Hybrid approaches like Agile-Waterfall and SAFe attempt to reconcile control with flexibility, making them attractive for large or regulated projects. Meanwhile, modern innovations such as DevOps, AI-driven SDLC, and low-code platforms emphasize automation, scalability, and faster delivery but often demand significant infrastructure changes and raise governance or ethical concerns. Overall, the table underscores the necessity for strategic SDM selection tailored to a project's size, complexity, regulatory demands, and team capabilities.

2. Critical Analysis

The evolution of software development methodologies is a direct response to organizational demands for faster delivery, improved product quality, and adaptable processes to meet dynamic requirements. Organizations increasingly embrace hybrid development models, which integrate structured control mechanisms with iterative, agile practices to support flexibility and responsiveness. This section organizes the research into four thematic areas, critically evaluating both practical implications and theoretical contributions. In the future, software development is expected to undergo major transformations through the integration of artificial intelligence (AI), quantum computing, and cognitive science. These technologies will allow organizations to make real-time, intelligent decisions, detect and resolve issues proactively, and model human problem-solving. Development environments shaped by cognitive behavior and user feedback could become autonomous, adaptive, and self-improving—leading to more efficient, secure, and customer-focused systems.

2.1 Modern Methodological Evolution and Shifts in Paradigms

Software development initially relied on structured models such as the Waterfall model, characterized by linear planning, sequential phases, and strict documentation (Royce, 1970). However, several scholars have criticized traditional models for their rigidity and inability to adapt to changes during development (Mahmoud, 2014; Ogundare & Osuolale, 2022).

JOURNAL OF EMERGING TECHNOLOGY AND DIGITAL TRANSFORMATION
ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

In contrast, the emergence of Agile methodologies in the early 2000s introduced iterative development, customer collaboration, and continuous delivery (Beck et al., 2001; Highsmith, 2009). Agile shifted focus toward adaptability and shortened feedback loops, enabling faster responses to customer needs and evolving project requirements. Sharma and Patidar (2024) further emphasized Agile's capacity to support continuous integration and improve inter-team coordination.

According to Dingsøyr et al. (2012, 2025), Agile has moved from niche applications in startups to widespread adoption in large enterprises. Their research demonstrates that Agile not only reshaped development processes but also redefined team structures and cultural dynamics, particularly in high-velocity environments.

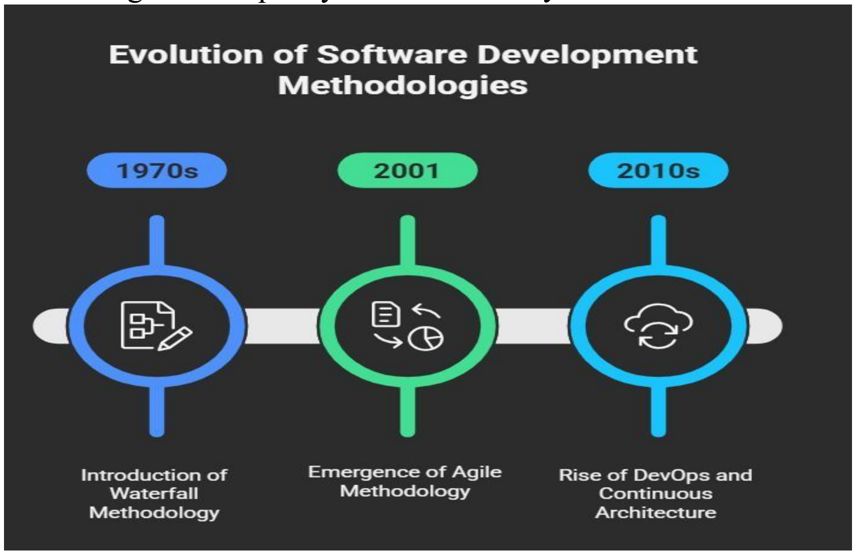

Evolution of Software Development Methodologies
1970s
2001
2010s
Introduction of Waterfall Methodology
Emergence of Agile Methodology
Rise of DevOps and Continuous Architecture

Fig2: SDM Year Wise

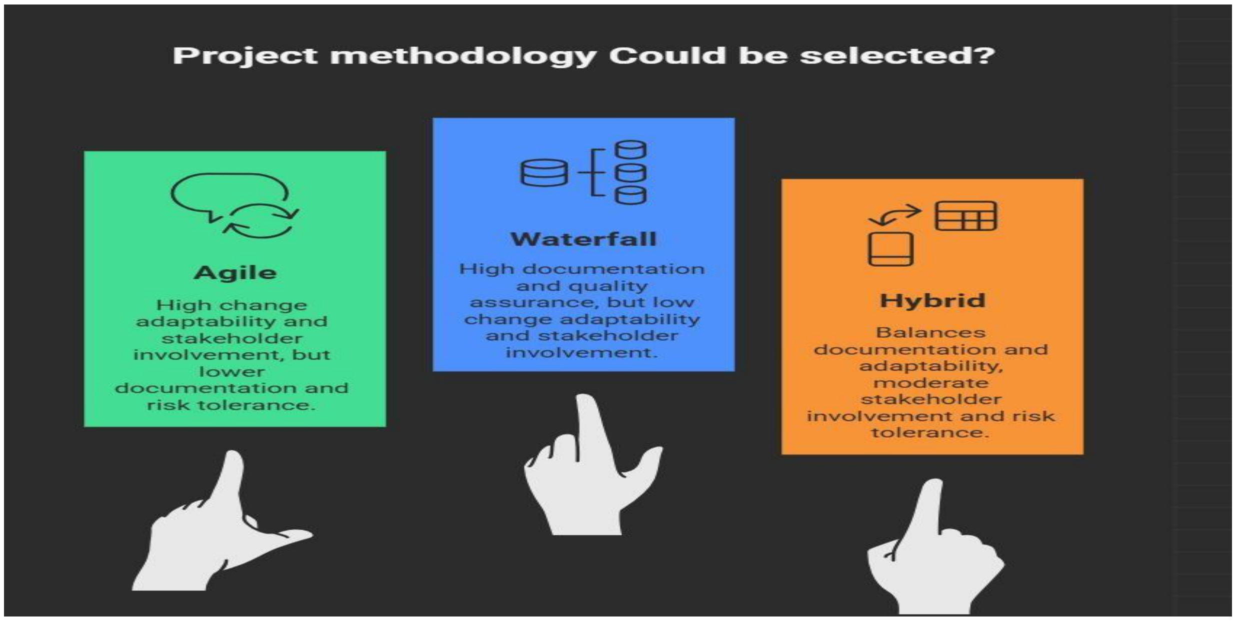
3.2 Trade-Offs and Methodology Recent Challenges

Agile and traditional models both have strengths and limitations. Tesoriero (2008) and Wohlin, Fors, and Wallin (2012) highlight ongoing tensions between rapid delivery and quality assurance. Agile often reduces emphasis on documentation and structured testing, while Waterfall emphasizes control and predictability but lacks agility in fast-changing environments.

Theocharis et al. (2021) argue for context-aware methodology selection based on factors like team expertise, risk, and stakeholder volatility. Baltes et al. (2024) introduce a structured framework for documenting trade-offs, enabling transparency and informed architectural decisions. Their model enhances decision-making by evaluating the relationship between benefits and costs. Furthermore, Hussain and Qamar (2022) warn against uncritical adoption of popular methods. They advocate customized models tailored to the project's regulatory demands, stakeholder needs, and technological constraints. This perspective supports the rising trend of hybrid or mixed-method approaches, especially in complex or compliance-heavy environments.

Fig 3 Challenges

3.3 Recent Hybrid and Integrated Methodologies


Project methodology Could be selected?
Agile
High change adaptability and stakeholder involvement, but lower documentation and risk tolerance.
Waterfall
High documentation and quality assurance, but low change adaptability and stakeholder involvement.
Hybrid
Balances documentation and adaptability, moderate stakeholder involvement and risk tolerance.

The inherent limitations of singular software development methodologies have led to the increasing adoption of hybrid approaches, particularly in projects that demand both structured oversight and iterative flexibility. Hybrid Agile methods combine traditional development practices with agile principles to accommodate the complexity of large-scale systems requiring both control and adaptability (Fitzgerald & Stol, 2017; Niazi et al., 2020). These approaches are especially beneficial when neither a fully Agile nor a

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

pure Waterfall model proves effective in isolation.

The evolution of Agile practices has led to further integration with DevOps methodologies, enabling continuous deployment and automation while preserving Agile's responsiveness. Balaji and Murugaiyan (2023), along with Rana (2023), demonstrate that incorporating DevOps practices—such as automation, continuous integration, and infrastructure as code—into Agile environments enhances delivery speed and improves fault response capabilities. Rather than compromising between approaches, these integrations aim to amplify their strengths. The synergy between Agile and DevOps results in accelerated workflows and heightened adaptability. Prakash et al. (2024) suggest that the inclusion of Cloud-native technologies creates a unified system where Agile, DevOps, and Cloud practices converge. This integration supports scalable, rapid development by overcoming traditional operational bottlenecks and reinforcing Agile's foundational values.

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

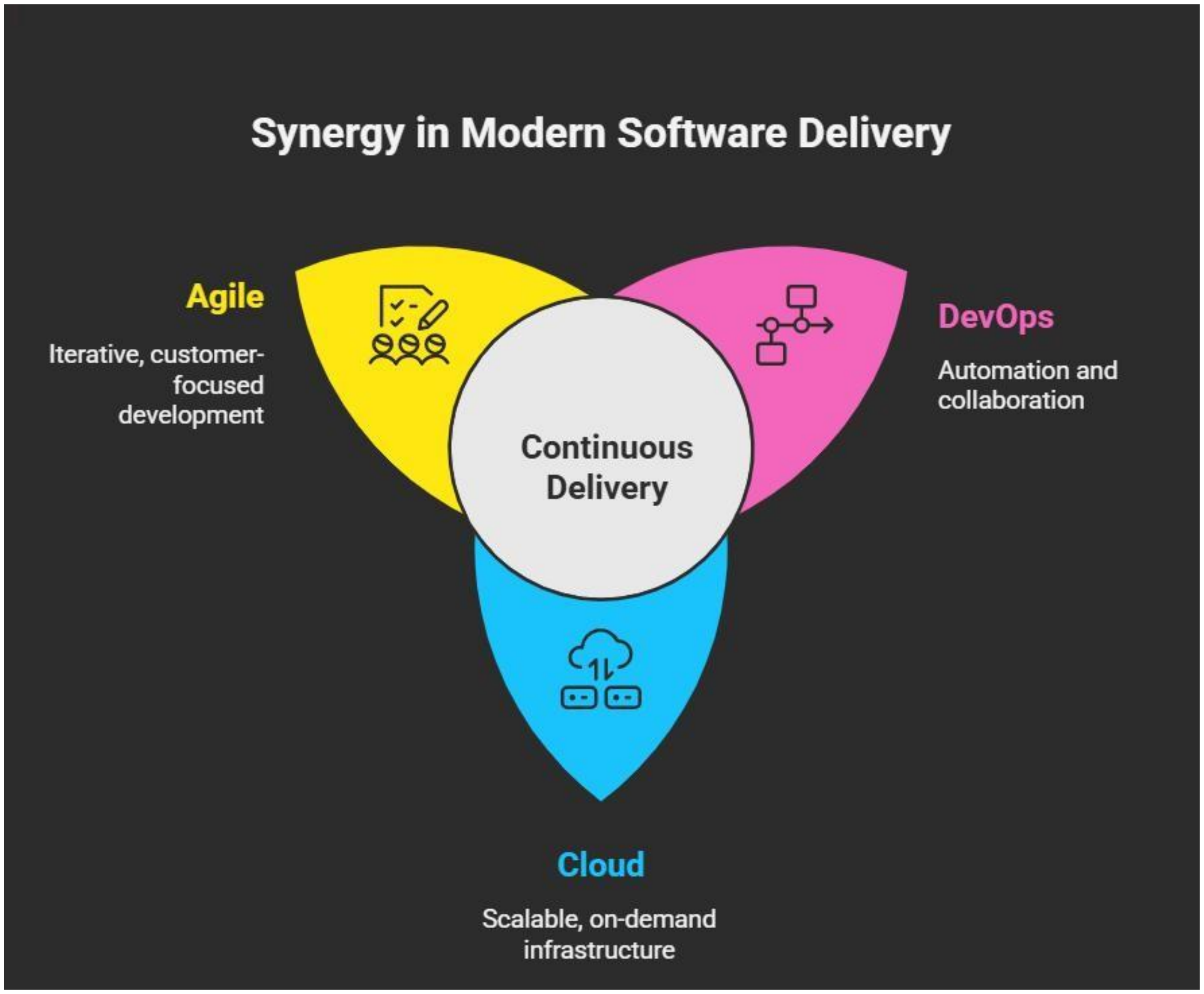

Synergy in Modern Software Delivery
Agile
Iterative, customer-focused development
DevOps
Automation and collaboration
Continuous Delivery
Cloud
Scalable, on-demand infrastructure

Fig 4 : Hybrid Technologies

3.4 Innovation, Quality, and Architectural Trade-offs

Software development methodology (SDM) professionals must prepare for an increasingly complex landscape in which system architectures are distributed, intelligent, and constantly evolving. Forecasting future trends requires integration of automated processes, intelligent systems, and predictive analytics to manage architectural, quality, and performance trade-offs in real-time (Kazman et al., 1999; Clements & Bass, 2003; Erder & Hirsh, 2021).

One proposed solution is the Quality Assurance Tradeoff Analysis Method

(QATAM), which helps evaluate the implications of various QA strategies on project outcomes (Beecham et al., 2003). QATAM supports data-informed decision-making by comparing the cost, impact, and timing of different quality assurance activities. However, Wohlin et al. (2012) caution that organizations often overlook how this quality practices align with their broader development frameworks. A structured trade-off analysis—using tools such as ATAM or QATAM—provides a strategic approach to quality planning, particularly in high-risk and large-scale projects.

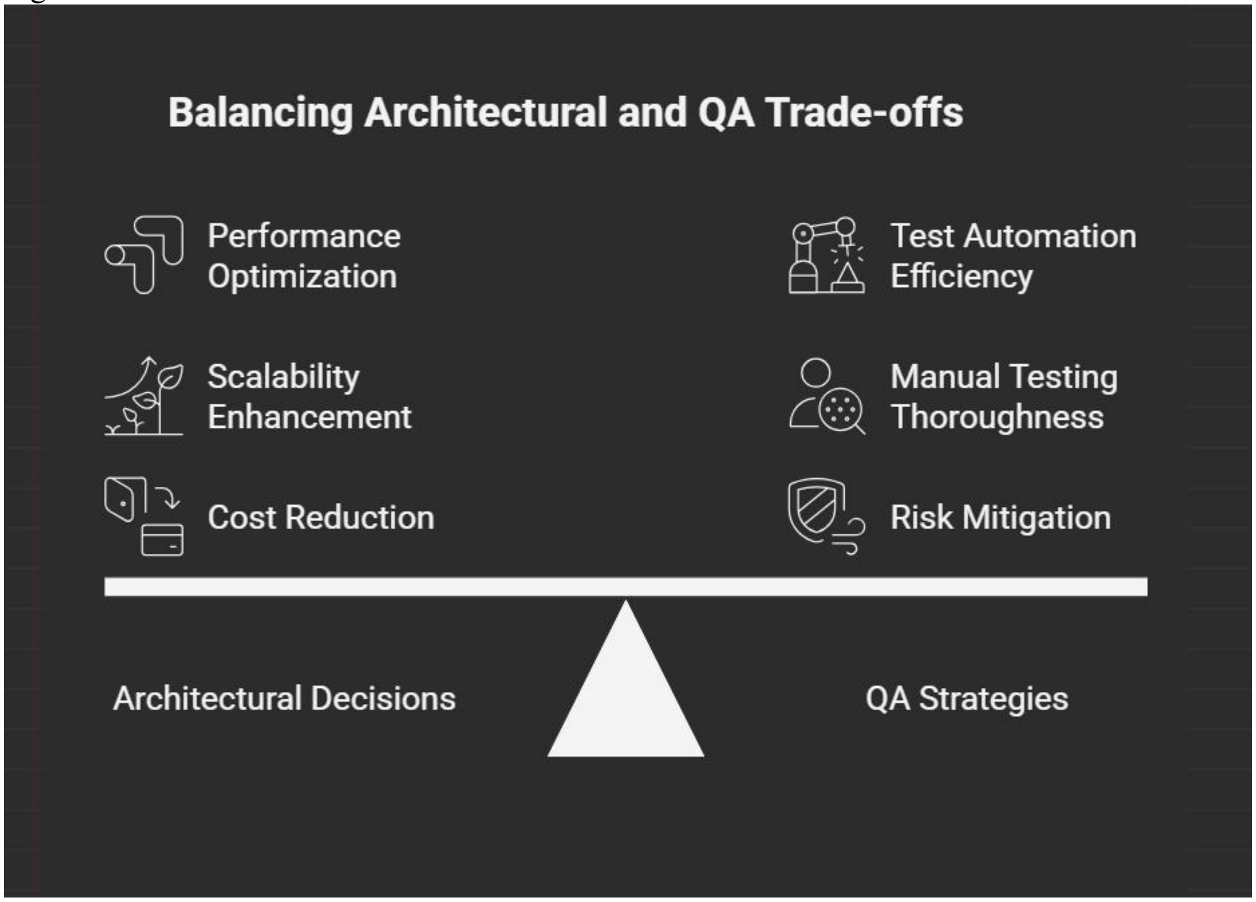


Fig 5 : QA tradeoffs

3. AI, Future Trends, and Methodological Disruption

Artificial Intelligence (AI) is poised to disrupt software development methodologies (SDMs) by moving beyond simple automation tools to become integrated

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

decision-making agents. According to Amershi et al. (2019), AI and machine learning (ML) are rapidly evolving into core SDM components that contribute to program design, automated testing, and intelligent decision support. These capabilities enable software teams to build, test, and adjust code with higher precision and adaptability.

Future SDMs must accommodate intelligent, self-adaptive systems capable of interpreting complex data streams and providing real-time recommendations. Zikria et al. (2023) note that the increasing complexity and distributed nature of software systems will demand adaptive learning architectures with dynamic feedback and automated risk mitigation. This implies a methodological shift toward cognitively inspired development environments, where AI mimics human behavior and feedback loops to self-improve continuously.

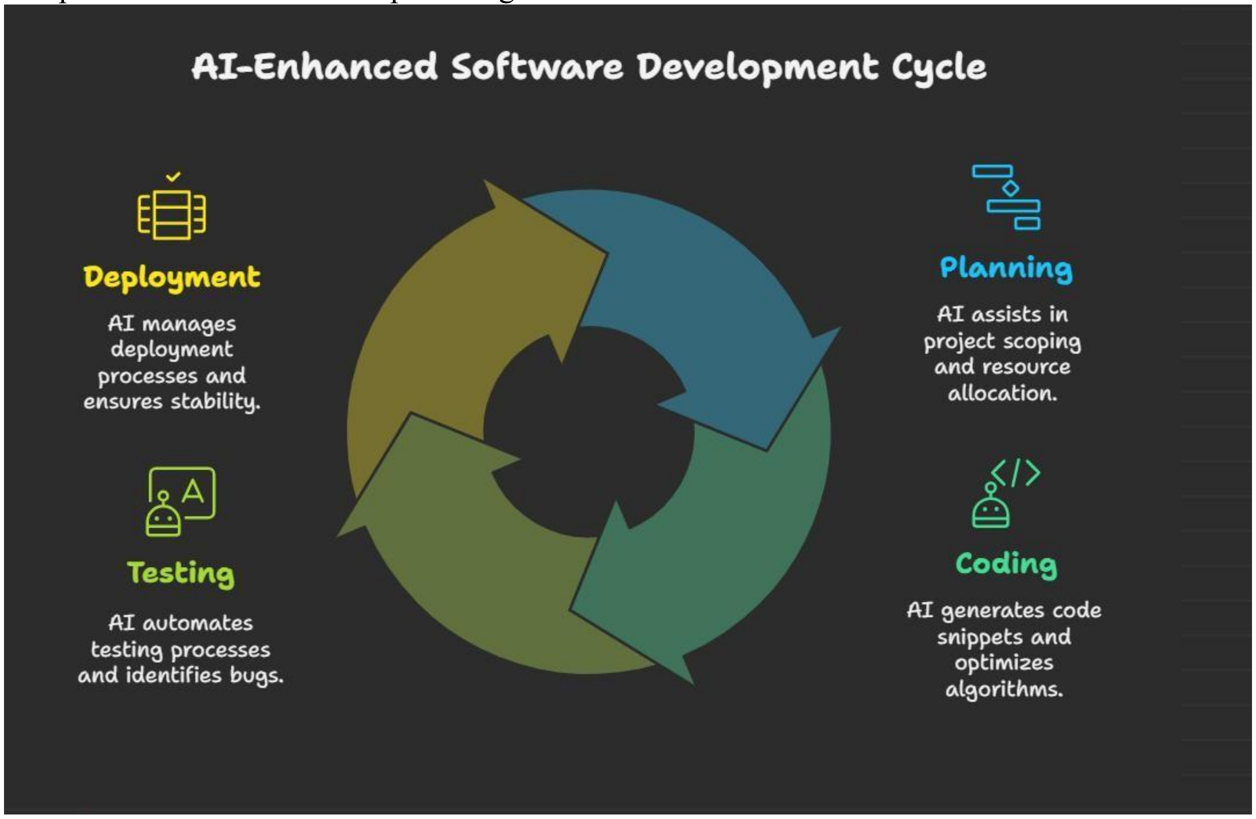

AI-Enhanced Software Development Cycle
Deployment
AI manages deployment processes and ensures stability.
Planning
AI assists in project scoping and resource allocation.
Testing
AI automates testing processes and identifies bugs.
Coding
AI generates code snippets and optimizes algorithms.

Fig 6 : AI based SDLC

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

4. Conclusion and Recommendations

The evolution of Software Development Methodologies (SDMs) reflects a dynamic shift from rigid, sequential approaches like Waterfall and V-Model toward more iterative and flexible models such as Agile. As software systems become more complex and organizations demand faster delivery, hybrid SDMs have emerged as an optimal solution—merging Agile's responsiveness with the structure of traditional methodologies. Modern frameworks such as DevOps and Scrumban exemplify this trend by enabling teams to maintain quality, compliance, and speed simultaneously. However, SDMs are not universally applicable; each approach has its contextual strengths and limitations. Waterfall remains suitable for stable, regulated environments, while Agile thrives in dynamic, iterative development cycles—provided there is high team maturity and communication. Hybrid methodologies offer a balanced path, supporting adaptability without losing oversight.

- Emerging trends such as AI integration, CI/CD pipelines, and cloud-native development are further reshaping SDM frameworks.
- Future SDMs must be modular, intelligent, and predictive—allowing organizations to customize processes, automate operations, and enhance decision-making accuracy.
- strategic selection and continuous refinement of methodologies will be essential for aligning SDM practices with specific project goals, team structures, and resource constraints.

5.1 Recommendations for Practitioners

Organizations should prioritize hybrid SDMs that combine the adaptability of Agile with the governance and documentation strengths of traditional models. Integrating DevOps into Agile practices can accelerate development cycles without compromising product quality. To remain competitive, organizations must invest in ongoing training and promote a culture of continuous improvement (e.g., Kaizen). Automated AI-supported tools should be implemented for testing, deployment, and feedback collection to reduce human error and accelerate iterations.

Before selecting an SDM, practitioners should analyze project-specific requirements, such as regulatory compliance, technical complexity, and team capabilities. For projects requiring extensive documentation and regulatory alignment, Waterfall or DevOps may be more appropriate than Agile. Moreover, successful implementation depends on strong leadership, effective communication channels, and empowered teams with decision-making autonomy and access to innovation.

5.2 Recommendations for Researchers

There is a critical need for longitudinal and empirical studies focused on hybrid SDMs. Researchers should explore how Agile principles function when embedded within traditional structures, especially in high-stakes or regulated industries. Studies should investigate how AI-driven tools influence project success, quality assurance, and risk mitigation. Special attention should be paid to low-code/no-code platforms, examining their impact on accessibility, scalability, and SDM compatibility.

Future research must adopt time-series analysis to evaluate long-term SDM performance in terms of delivery speed, customer satisfaction, and team cohesion. Additionally, a comprehensive framework for hybrid SDMs should be developed to guide organizations in selecting and customizing methodologies that align with their specific operational needs, regulatory environments, and strategic objectives.

5.3 Final Thoughts

SDM development is a continuous and evolving effort aimed at optimizing how software is designed, delivered, and maintained. With advancements in AI, automation, and cloud computing, the next generation of SDMs must be adaptive, predictive, and human-centric. The success of any SDM lies in its alignment with organizational goals, team capabilities, and external demands. This review provides both a strategic outlook and a practical roadmap for organizations and researchers navigating the future of software development.

6. REFERENCES

Zikria, Y. B., et al. (2023). A Comprehensive Review of Software Development Life Cycle Methodologies: gPros, Cons, and Future Directions. ResearchGate. https://www.researchgate.net/publication/379652502

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718

VOLUME . 4 ISSUE . 2 (2025)

Balaji, S., & Murugaiyan, M. S. (2023). DevOps Enabled Agile: Combining Agile and DevOps Methodologies for Software Development. ResearchGate. https://www.researchgate.net/publication/365970960

3. Ogundare, S., & Osuolale, O. (2022). A Comparative Case Study of Waterfall and Agile vManagement. SAR Journal, 5(1), 52–62. https://www.sarjournal.com/content/51/SARJournalMarch2022_52_62.pdf

Hardgrave, B. C. (2017). A Study of Software Development Methodologies. University of Arkansas. https://scholarworks.uark.edu/cgi/viewcontent.cgi?article=1105&context=csceuht

Baltes, S., et al. (2024). Communicating Study Design Trade-offs in Software Engineering. ACM Transactions on Software Engineering and Methodology. https://dl.acm.org/doi/10.1145/3649598

Technology Buzz. (2023). Software Development Methodologies and Trends 2023–24. Medium. https://technlogy-buzz.medium.com/software-development-methodologies-and- trends-2023-24-1fd7dd0335e1

Theocharis, G., et al. (2021). Agile versus Waterfall Project Management: Decision Model for Selecting the Appropriate Approach. Procedia Computer Science, 181, 231–238. https://www.sciencedirect.com/science/article/pii/S1877050921002702

8. Mahmoud, Q. H. (2014). A Comparative Overview of the Evolution of Software Development Models. ResearchGate. https://www.researchgate.net/publication/267711880

9. Niazi, M., et al. (2020). An Insight into Hybrid Agile Software Development Approaches. JATIT, 102(2), 47–54. https://www.jatit.org/volumes/Vol102No2/6Vol102No2.pdf

10. Hussain, S., & Qamar, U. (2022). A Comparative Study of Agile and Waterfall Software Development Methodologies. ResearchGate. https://www.researchgate.net/publication/361872079

11. Sharma, M., & Patidar, N. (2024). A Systematic Review of Software Development Methodologies and Their Trends. JRTCSE.

ONLINE ISSN 3006-9726
PRINT ISSN 3006-9718
VOLUME . 4 ISSUE . 2 (2025)

https://jrtcse.com/index.php/home/article/view/JRTCSE.2024.5.7

12. Tesoriero, R. (2008). Trade-offs Between Productivity and Quality in Selecting Software Development Practices. ResearchGate. https://www.researchgate.net/publication/3248028

13. Dingsøyr, T., et al. (2025). A Systematic Literature Review of Agile Software Development: State of Research and Future Directions. Information & Software Technology. https://www.sciencedirect.com/science/article/abs/pii/S0950584925000667

14. Rana, A. (2023). Integrating DevOps with Agile and Other Software Development Methodologies. ResearchGate. https://www.researchgate.net/publication/382852396

15. Misra, S. C., & Kumar, V. (2023). Structured Software Development Versus Agile Software Development. Springer. https://link.springer.com/article/10.1007/s13198-023-01958-5

16. OfficeTimeline. (2023). Software Development Methodologies Timeline. https://www.officetimeline.com/blog/software-development-methodologies-timeline

17. Wohlin, C., et al. (2012). Software Quality Trade-offs: A Systematic Map. Information & Software Technology, 54(7), 743–758. https://www.sciencedirect.com/science/article/abs/pii/S0950584912000195

18. Concise Software. (2023). The Most Common Software Development Methodologies in 2023. https://concisesoftware.com/blog/the-most-common-software-development-methodologies-in-2023

19. Prakash, P., et al. (2024). A Systematic Literature Review on Agile, Cloud, and DevOps Integration. ACM/Elsevier. https://dl.acm.org/doi/10.1016/j.infsof.2024.107569

20. JSAER. (2022). Comparing Agile and Waterfall Methodologies: A Strategic Evaluation. JSAER, 9(9), 108–111. https://jsaer.com/download/vol-9-iss-9-2022/JSAER2022-9-9- 108-111.pdf

21. Intetics. (2023). A Brief History of Software Development Methodologies.

https://intetics.com/blog/a-brief-history-of-software-development-methodologies

22. Beecham, S., et al. (2003). Software Process Improvement Initiatives Based on Quality Assurance Strategies: A QATAM Pilot Application. ResearchGate. https://www.researchgate.net/publication/221045890

23. Kazman, R., et al. (1999). Architecture Tradeoff Analysis Method (ATAM): Evaluating Software Architectures. SEI/CMU. https://www.geeksforgeeks.org/architecture-tradeoff- analysis-method-atam

24. Clements, P., & Bass, L. (2003). Quality Trade-off Analysis: The ATAM Method. GlobalSpec. https://www.globalspec.com/reference/39451/203279

25. Erder, M., & Hirsh, P. (2021). Validating the Architecture: A Brief Overview of the ArchitectureTradeoff Analysis Method. Continuous Architecture. https://www.sciencedirect.com/topics/computer-science/architecture-tradeoff-analysis-method

26. Highsmith, J. (2009). Agile Software Development: The Business of Innovation. Addison- Wesley.

27. Bass, L., & Weber, I. (2015). The Impact of DevOps on Software Development Efficiency. ACM Queue.

28. Fitzgerald, B., & Stol, K.-J. (2017). Hybrid Software Development Approaches: Merging Agile and Waterfall. Journal of Systems and Software, 133, 68–82.

29. Poppendieck, M., & Cusumano, M. (2012). The Role of Lean Principles in Software Development. IEEE Software, 29(5), 26–32.

30. Amershi, S., et al. (2019). Future Directions in Software Development Methodologies: Embracing AI and Machine Learning. Communications of the ACM, 62(9), 62–71.

31. Royce, W. W. (1970). Managing the development of large software systems. Proceedings of IEEE WESCON.

32. Beck, K., et al. (2001). Manifesto for Agile Software Development.

33. Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718
VOLUME . 4 ISSUE . 2 (2025)

software development. Journal of Systems and Software, 85(6), 1213–1221.

34. Jalali, S., & Wohlin, C. (2012). Agile practices in global software engineering—A systematic map. 2012 International Conference on Global Software Engineering.

35. Fitzgerald, B., & Stol, K. J. (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 123, 176–189.

ONLINE ISSN
3006-9726
PRINT ISSN
3006-9718

VOLUME . 4 ISSUE . 2 (2025)