Information Technology Studies Journal (ITECH)
Vol 3 (2) 2026 : 218-228

ARTIFICIAL INTELLIGENCE DRIVEN SOFTWARE ENGINEERING: CURRENT DEVELOPMENTS, CHALLENGES, AND FUTURE RESEARCH DIRECTIONS

REKAYASA PERANGKAT LUNAK BERBASIS KECERDASAN BUATAN: PERKEMBANGAN TERKINI, TANTANGAN, DAN ARAH PENELITIAN MASA DEPAN

Achmad Ardiansyah[1], Mepa Kurniasih[2]
Universitas Budi Luhur, Jakarta, Indonesia[1,2]
*ahd.ardiansyah@gmail.com, mepa.kurnia@gmail.com

**Corresponding Author*

ABSTRACT
The convergence of Artificial Intelligence (AI) and Software Engineering (SE) has fundamentally reshaped how software systems are conceived, designed, built, tested, and maintained. Following the widespread commercial deployment of Large Language Models (LLMs) and generative AI tools such as GitHub Copilot, this paper presents a narrative review of current developments, key challenges, and future research directions in AI-driven Software Engineering (AI4SE). Drawing on peer-reviewed systematic literature reviews, empirical studies, and industry survey data published between 2019 and 2025, this review synthesizes evidence across the software development life cycle (SDLC), including requirements engineering, software design and architecture, automated code generation, software testing, and software maintenance. The review shows that AI adoption among practitioners has grown rapidly. Industry surveys report that more than 80% of developers now use AI tools in their workflow while empirical studies report productivity gains of up to 55% for specific coding tasks. However, the literature also converges on persistent challenges, including code quality and technical debt, explainability and trust, data and model bias, security vulnerabilities in AI-generated code, and organizational barriers to adoption. This review proposes an integrative conceptual framework that maps AI capabilities onto SDLC phases and challenge dimensions, and outlines an agenda for future research emphasizing trustworthy AI4SE, human-AI collaboration models, and empirical validation at scale. The paper is intended to serve as a reference synthesis for researchers, practitioners, and policymakers navigating the rapidly evolving intersection of AI and software engineering.
Keywords: artificial intelligence; software engineering; large language models; generative AI; narrative review; software development life cycle

ABSTRAK
Konvergensi antara Artificial Intelligence (AI) dan Software Engineering (SE) telah mengubah secara fundamental cara sistem perangkat lunak dikonseptualisasikan, dirancang, dibangun, diuji, dan dipelihara. Seiring dengan meluasnya penerapan komersial Large Language Models (LLMs) dan alat generative AI seperti GitHub Copilot, artikel ini menyajikan tinjauan naratif mengenai perkembangan terkini, tantangan utama, dan arah penelitian masa depan dalam Rekayasa Perangkat Lunak Berbasis Kecerdasan Buatan (AI-Driven Software Engineering/AI4SE). Berdasarkan tinjauan literatur sistematis yang telah ditelaah sejawat, studi empiris, dan data survei industri yang dipublikasikan pada periode 2019–2025, artikel ini mensintesis bukti pada seluruh tahapan Software Development Life Cycle (SDLC), yang meliputi rekayasa kebutuhan (requirements engineering), desain dan arsitektur perangkat lunak, pembangkitan kode secara otomatis, pengujian perangkat lunak, serta pemeliharaan perangkat lunak. Hasil tinjauan menunjukkan bahwa adopsi AI di kalangan praktisi berkembang sangat pesat. Berbagai survei industri melaporkan bahwa lebih dari 80% pengembang perangkat lunak kini menggunakan alat AI dalam alur kerja mereka, sementara studi empiris menunjukkan peningkatan produktivitas hingga 55% pada tugas-tugas pengkodean tertentu. Namun demikian, literatur juga secara konsisten mengidentifikasi sejumlah tantangan yang masih berlanjut, termasuk kualitas kode dan utang teknis (technical debt), kemampuan penjelasan (explainability) dan kepercayaan (trust), bias data dan model, kerentanan keamanan pada kode yang dihasilkan AI, serta hambatan organisasi dalam proses adopsi. Tinjauan ini mengusulkan sebuah kerangka konseptual integratif yang memetakan kemampuan AI ke

dalam setiap fase SDLC beserta dimensi tantangannya, sekaligus merumuskan agenda penelitian masa depan yang menekankan pengembangan AI4SE yang dapat dipercaya (trustworthy AI4SE), model kolaborasi manusia–AI, serta validasi empiris dalam skala besar. Artikel ini diharapkan menjadi referensi komprehensif bagi peneliti, praktisi, dan pembuat kebijakan dalam memahami serta mengembangkan bidang rekayasa perangkat lunak yang didukung kecerdasan buatan yang terus berkembang dengan cepat.

Kata Kunci: kecerdasan buatan; rekayasa perangkat lunak; large language models; AI generatif; tinjauan naratif; siklus hidup pengembangan perangkat lunak.

1. INTRODUCTION

Software Engineering (SE) has historically evolved through successive waves of methodological and technological transformation from structured programming, to object-oriented design, to agile and DevOps practices. The current wave is driven by Artificial Intelligence (AI), and in particular by advances in machine learning (ML), deep learning, and, most recently, Large Language Models (LLMs) capable of understanding and generating source code, natural language requirements, and test artifacts (Hou et al., 2024). Nascimento et al. (2020) note that as AI/ML systems became widely adopted as core value propositions across industries, software engineering itself had to adapt, giving rise to a distinct sub-discipline concerned with engineering AI-based systems and, reciprocally, applying AI techniques to improve software engineering practice.

The scale of this shift is reflected in adoption statistics from both industry and developer communities. According to the Stack Overflow (2024) Developer Survey, which collected responses from more than 65,000 developers worldwide, 76% of respondents reported using or planning to use AI tools in their development process, up from 70% the previous year, and 82% of developers who use AI tools reported using them specifically to write code (Stack Overflow, 2024). By the 2025 edition of the same survey, AI tool usage had climbed further to 84% of respondents who use or plan to use AI tools, even as trust in the accuracy of AI-generated output declined, with 46% of developers reporting distrust of AI output compared to 31% a year earlier (Stack Overflow, 2025). This apparent paradox rising adoption alongside declining trust signals that AI is being normalized as a co-worker in the development process even as its limitations become more visible to practitioners.

Complementary evidence comes from controlled experiments and telemetry-based studies of GitHub Copilot, one of the most widely deployed AI pair-programming tools. GitHub (2024) reported that in a controlled experiment, developers who used Copilot to implement an HTTP server in JavaScript completed the task 55% faster than a control group, and that developers using the tool reported it helped them stay “in the flow” (73%) and conserve mental effort during repetitive tasks (87%). By mid-2025, GitHub Copilot had reportedly reached approximately 20 million cumulative users and had been adopted by roughly 90% of Fortune 100 companies (Getpanto, 2025), illustrating the speed at which generative AI coding assistants have moved from research prototypes to enterprise-scale infrastructure.

At the research level, this rapid practical adoption has been mirrored by an equally rapid expansion of the scholarly literature. Hou et al. (2024) conducted a systematic literature review of LLMs applied to software engineering (LLM4SE) and identified 395 primary studies published between January 2017 and January 2024 alone, spanning tasks from requirements analysis and code generation to program repair and software maintenance. Similarly, a decade-spanning systematic review of AI integration across SE phases and activities from 2013 to 2023 selected 110 primary studies, reporting an intensifying intersection between the two fields across planning, requirements engineering, design, development, testing, deployment, and maintenance activities. This exponential growth in both practical adoption and academic output motivates a synthesis that maps what is currently known, what challenges remain unresolved, and where future research should be directed.

Despite the proliferation of individual empirical studies and phase-specific systematic reviews, the AI4SE literature remains fragmented across sub-domains (e.g., AI for testing, AI for requirements engineering, AI for code generation) and across research communities (software engineering venues versus AI/ML venues). Existing systematic reviews tend to focus narrowly on a single SDLC phase (e.g., Bucaioni et al.'s review of AI in software architecture, or reviews focused specifically on LLM-based test case generation) or on a single technique family (e.g., generative AI specifically, rather than AI broadly). There remains a need for a synthesis that (i) integrates evidence across the entire SDLC, (ii) juxtaposes technical developments against the socio-technical challenges reported by practitioners, and (iii) distills a coherent future research agenda from the union of these fragmented reviews. This narrative review addresses that gap.

This review is guided by three research questions:

RQ1. How has AI been applied across the different phases of the Software Development Life Cycle (SDLC), and what is the current state of the art?

RQ2. What are the principal technical, ethical, and organizational challenges reported in the literature regarding the implementation of AI-driven software engineering?

RQ3. What research gaps exist in the current literature, and what future research directions can be identified for AI4SE?

The objectives of this review are threefold: first, to synthesize current developments in AI4SE across the SDLC based on recent systematic reviews and empirical studies; second, to consolidate the challenges reported across technical, ethical, and organizational dimensions into an integrative framework; and third, to propose a future research agenda grounded in the identified gaps. The contribution of this paper is a narrative synthesis and an integrative conceptual framework (presented in Section 4) that can serve as an orientation map for researchers entering the field and as a reference point for practitioners evaluating AI adoption strategies.

The remainder of this paper is organized following the IMRAD structure. Section 2 describes the narrative review methodology, including the search strategy and study selection approach. Section 3 presents the results, organized by SDLC phase. Section 4 discusses the synthesized challenges, presents the integrative framework and accompanying diagrams, and outlines future research directions. Section 5 concludes the paper.

2. METHOD

2.1 Review Approach

This paper adopts a narrative review approach rather than a systematic literature review (SLR). A narrative review is appropriate when the objective is to provide a broad, integrative synthesis of a rapidly evolving, cross-disciplinary topic, drawing on existing systematic reviews, empirical studies, and industry reports to construct a coherent conceptual picture, rather than to exhaustively enumerate and statistically aggregate every primary study (Kitchenham et al., 2009). Given that AI4SE already possesses several high-quality systematic reviews at the sub-domain level (e.g., LLM4SE, AI for software architecture, AI for software testing), this review adopts a "review-of-reviews and recent primary-studies" strategy, synthesizing findings from these existing systematic works alongside recent (2023–2025) primary empirical studies and authoritative industry survey data.

2.2 Search Strategy

A structured search was conducted across Scopus-indexed and widely recognized academic databases, namely IEEE Xplore, ACM Digital Library, ScienceDirect, SpringerLink, Wiley Online Library, and arXiv (as a preprint source for rapidly evolving LLM research), supplemented by Google Scholar for cross-verification of citation counts and venue quality. The search string combined the following term clusters using Boolean operators: ("artificial intelligence" OR "machine learning" OR "large language model*" OR "generative AI") AND

("software engineering" OR "software development") AND ("review" OR "survey" OR "systematic literature review" OR "empirical study"). The search was restricted to English-language publications from 2019 to 2025, reflecting the period of most intensive LLM-driven transformation in the field, while foundational methodological references (e.g., Kitchenham et al., 2009 on SLR guidelines) were retained regardless of date due to their continued methodological relevance.

2.3 Inclusion and Exclusion Criteria

Studies were included if they (i) were peer-reviewed journal articles, conference papers, or reputable industry survey reports; (ii) addressed the application of AI/ML/LLM techniques to at least one phase of the software development life cycle; and (iii) reported empirical findings, a systematic synthesis, or quantitative survey data. Studies were excluded if they (i) were not accessible in full text, (ii) focused exclusively on AI as a subject of software engineering (i.e., building AI systems) without addressing AI as a tool for engineering software, or (iii) lacked methodological transparency (e.g., blog posts without a documented data source). Priority was given to systematic literature reviews and studies published in Scopus-indexed Q1 journals and top-tier SE venues (ICSE, FSE, ASE, ISSTA, TSE, TOSEM, EMSE) to ensure methodological rigor of the underlying evidence base.

2.4 Synthesis Approach

Given the narrative review design, synthesis followed a thematic-integrative approach: findings from the selected sources were organized according to (a) SDLC phase (requirements, design/architecture, coding, testing, maintenance/DevOps) and (b) cross-cutting challenge dimension (technical, ethical, organizational). This thematic structure, rather than a chronological or purely bibliometric structure, was chosen to directly address the three research questions and to support the construction of the integrative framework presented in Section 4.

3. RESULTS

3.1 Overview of the Evidence Base

The synthesized evidence base spans systematic reviews covering different time windows and scopes: a review of software engineering practices for AI/ML systems covering 1990–2019 (Nascimento et al., 2020); a decade-focused review of AI integration across SE phases from 2013–2023 covering 110 primary studies; a review of generative AI in software engineering process optimization covering 117 studies; a large-scale LLM4SE systematic review covering 395 primary studies from 2017–2024 (Hou et al., 2024); a review of AI support for software architecture practice covering 51 primary studies; a systematic review of software quality for AI-based software from 1988–2020; and a 2025 review following PRISMA guidelines covering 135 peer-reviewed papers from 2010–2025 on AI innovations across SE. Collectively, these reviews encompass several hundred unique primary studies, providing a broad empirical foundation for the synthesis that follows.

3.2 AI in Requirements Engineering

AI, and particularly Natural Language Processing (NLP) and more recently LLMs, has been applied to requirements elicitation, classification, and analysis. Hou et al. (2024) identify requirements analysis as one of the SE tasks most influenced by LLMs, noting their use in analyzing textual requirements originating from diverse stakeholders during both construction and maintenance activities. Automated classification of functional versus non-functional requirements, ambiguity detection in natural-language requirement documents, and requirements traceability are recurring application areas identified across the broader SE-AI literature. These capabilities primarily target the reduction of manual analyst effort and the

earlier detection of requirement defects, before they propagate into design and implementation.

3.3 AI in Software Design and Architecture

Bucaioni et al. (2025) conducted a systematic review of 51 peer-reviewed primary studies examining AI's role in software architecture practice, systematically mapping AI contributions onto 17 practitioner-reported software architecture challenges derived from empirical interviews. Their review reports that architectural practices which rely on complex trade-off analysis, documentation, and long-term evolution remain traditionally manual, error-prone, and difficult to sustain, and that AI techniques are increasingly explored to support architectural decision-making, pattern recognition, and documentation generation, although the review concludes that AI's explicit role in this phase remains "insufficiently understood" relative to its role in coding and testing, indicating design and architecture as a comparatively under-researched SDLC phase.

3.4 AI in Code Generation and Programming Assistance

Code generation is the most mature and most extensively studied application of AI in software engineering, driven primarily by transformer-based LLMs. Hou et al. (2024) report that code generation and related code artifacts constitute a dominant theme within the 395 studies reviewed. At the tool level, GitHub Copilot has become the most widely deployed instance of this technology in industry. GitHub's own controlled experiment found that developers implementing an HTTP server in JavaScript completed the task 55% faster with Copilot than without it (GitHub, 2024), while a broader survey of Copilot users found that 88% of developers reported feeling more productive when using the tool (DevOps.com, 2024). Industry-compiled statistics further indicate that Copilot-generated code constitutes a substantial share of code committed by its users, with code acceptance rates in the range of 27–30% of suggestions offered, and that a majority of accepted suggestions are retained in final code submissions (Getpanto, 2025), suggesting that AI-generated code is increasingly production-oriented rather than purely exploratory. At the same time, the Stack Overflow (2025) survey found that developer trust in the accuracy of AI-generated output has declined even as usage has increased, and that developers show particular resistance to applying AI in high-responsibility tasks such as deployment and monitoring (76% do not plan to use AI for this) and project planning (69% do not plan to), indicating that adoption of AI for code generation has significantly outpaced adoption for higher-stakes, systemic engineering decisions.

3.5 AI in Software Testing and Quality Assurance

Software testing has emerged as a particularly active sub-domain for LLM-based automation. A recent MDPI review focused specifically on LLMs for automated test case generation, synthesizing studies that evaluate LLM effectiveness relative to traditional search-based and symbolic-execution-based test generation tools, and highlighting both the opportunities (context-aware test generation informed by natural-language understanding of code) and the persistent challenges (test correctness, coverage adequacy, and oracle generation) associated with this approach. A large-scale empirical evaluation published in IEEE Transactions on Software Engineering assessed LLM-based automated unit test generation directly against established baselines, finding that while LLMs can generate plausible and often executable test cases, questions of test quality, redundancy, and fault-detection effectiveness remain open research problems. Wang et al. (2024), as cited within the broader LLM4SE literature, similarly frame software testing with LLMs as an emerging "survey, landscape, and vision" area rather than a solved problem, underscoring that testing remains one of the more empirically contested application areas of AI4SE despite its popularity as a research topic.

3.6 AI in Software Maintenance, DevOps, and Software Quality

Beyond initial construction, AI techniques have been applied to program repair, code review assistance, defect prediction, and software quality assessment for AI-based systems themselves. A systematic literature review on software quality for AI-based software, published in *Empirical Software Engineering*, examined quality attributes, applied models, challenges, and practices reported in the literature from 1988 to 2020, finding that AI-based software quality differs meaningfully from traditional software quality because AI systems address more complex and less deterministic problem classes, complicating the direct transfer of traditional SE quality assurance practices. This finding is echoed in the broader literature on engineering AI-powered systems, which frequently notes that traditional software engineering taxonomies of quality, testability, and maintainability require substantial adaptation when applied to ML-based components (Lwakatare et al., 2019).

3.7 Synthesis Table: AI Applications Across the SDLC

The table below summarizes representative AI techniques, primary objectives, and reported outcomes across SDLC phases, synthesized from the sources reviewed above.

SDLC Phase	Representative AI Techniques	Primary Objective	Reported Outcome / Evidence
Requirements Engineering	NLP, text classification, LLM-based analysis	Requirement classification, ambiguity detection, traceability	Reduced manual analyst effort (Hou et al., 2024)
Design & Architecture	ML-based pattern recognition, LLM-assisted documentation	Architectural decision support, trade-off analysis	Coverage of 17 practitioner-reported challenges; role still under-researched (Bucaioni et al., 2025)
Coding / Implementation	Transformer-based LLMs (e.g., Copilot, Codex-family models)	Code completion, generation, pair-programming	Up to 55% faster task completion (GitHub, 2024); 88% report higher productivity (DevOps.com, 2024)
Testing & QA	LLM-based test generation, search-based testing hybrids	Automated unit/integration test generation	Promising but variable test quality and fault-detection effectiveness (IEEE TSE, 2023/2024 evaluations)
Maintenance / DevOps	AI-assisted code review, defect prediction, program repair	Early defect detection, maintainability support	Traditional SE quality models require adaptation for AI-based components (EMSE, 2021)

4. DISCUSSION

4.1 Cross-Cutting Challenges

Synthesizing across the sources reviewed in Section 3, three broad categories of challenge recur: technical, ethical/trust-related, and organizational challenges.

Technical challenges include code quality and technical debt introduced by AI-generated code, limited semantic understanding of large or unfamiliar codebases, integration difficulties between AI tools and existing toolchains, and unresolved questions of test adequacy and fault-detection effectiveness for AI-generated tests. The IET Software review by Ali (2025), synthesizing 135 peer-reviewed papers under PRISMA guidelines, frames these as

central “innovations, challenges, and future directions” spanning the full lifecycle, reinforcing that technical maturity varies substantially by SDLC phase, with code generation and testing considerably more mature than architecture-level and requirements-level applications.

Ethical and trust-related challenges center on explainability, bias, misinformation risk, and intellectual property concerns in AI-generated artifacts. The Stack Overflow (2024) survey found that misinformation and disinformation in AI results were the top ethical concern for 79% of surveyed developers, followed by concerns over source attribution (65%). The subsequent 2025 survey documented a marked decline in developer trust in AI output accuracy from roughly 40% trusting AI accuracy in earlier years to approximately 29–33% by 2025 even as usage continued to climb, illustrating a widening gap between adoption and confidence that has direct implications for responsible-use governance in engineering organizations.

Organizational challenges include the skills gap required to effectively supervise and validate AI-generated artifacts, resistance to workflow change, governance and accountability structures for AI-assisted decisions, and the uneven distribution of AI adoption across high-risk versus low-risk engineering tasks. As noted in Section 3.4, developers report strong reluctance to delegate high-responsibility tasks such as deployment, monitoring, and project planning to AI tools, suggesting that organizational trust calibration rather than technical capability alone is currently a binding constraint on deeper AI integration into the SDLC.

4.2 Integrative Conceptual Framework

Based on the synthesis above, this review proposes an integrative framework the AI4SE Adoption and Impact Framework that connects AI techniques, SDLC phases, and challenge dimensions into a single conceptual structure. The framework is presented textually below as a layered diagram (Figure 1) and further decomposed into a maturity-versus-risk mapping (Figure 2).

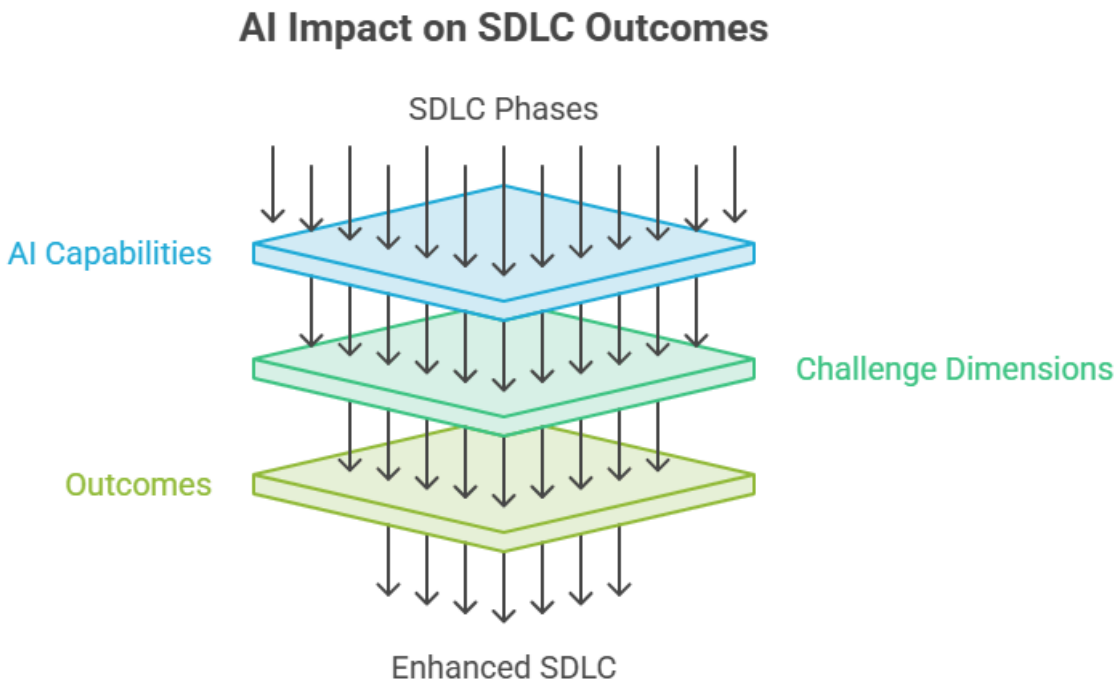

Figure 1. AI4SE Adoption and Impact Framework (layered structure)

The framework reads bottom-up: AI capabilities (Layer 2) are applied within specific SDLC phases (Layer 1); the effectiveness of that application is moderated by technical, ethical, and organizational challenge dimensions (Layer 3); and the net result manifests as observable outcomes (Layer 4) such as productivity gains, variable quality, and critically a widening trust gap. This layered structure explains the apparent paradox identified in Section 1.1: adoption (Layers 1–2) can rise even while trust (Layer 4) falls, because the moderating challenge layer (Layer 3) is not resolving as quickly as capability is diffusing.

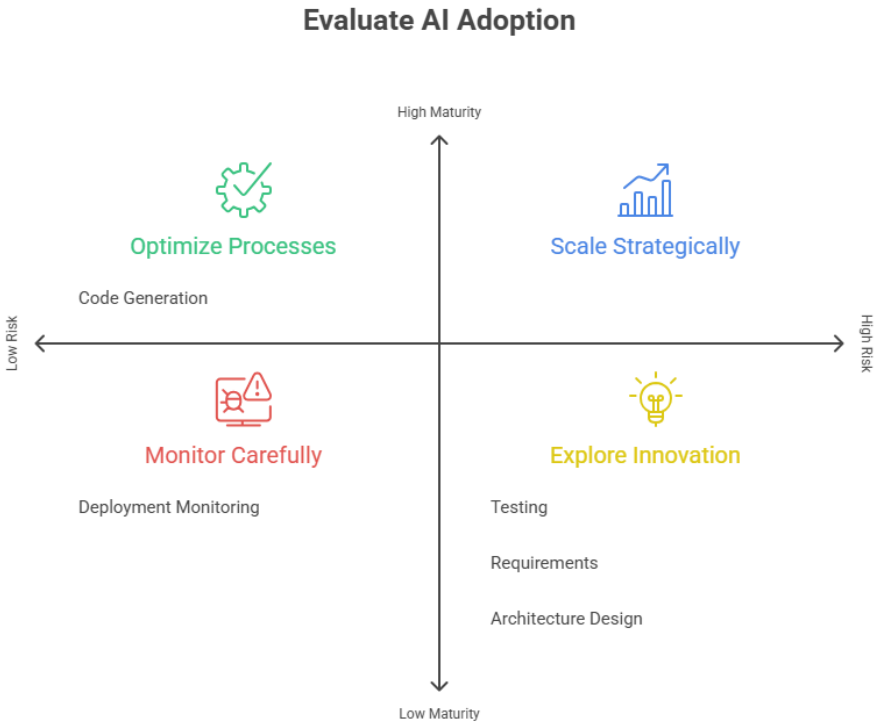

Figure 2. Maturity-versus-Risk Mapping of AI4SE Application Areas

Figure 2 illustrates a pattern consistent with the evidence in Section 3 and Section 4.1: AI maturity and adoption are currently concentrated in lower-stakes, well-bounded tasks (code completion, unit test generation), while high-stakes, systemic tasks (architecture decisions, deployment, monitoring) show both lower AI maturity and lower practitioner willingness to delegate, consistent with the Stack Overflow (2025) finding that 76% of developers do not plan to use AI for deployment and monitoring tasks.

4.3 Answering the Research Questions

RQ1 (Current developments across the SDLC): The evidence indicates uneven but accelerating AI penetration across the SDLC. Code generation is the most mature and most widely adopted application, supported by strong empirical productivity evidence (GitHub, 2024; DevOps.com, 2024). Testing is a fast-growing but still empirically contested area, with open questions about test adequacy. Requirements engineering and software architecture remain comparatively early-stage, with Bucaioni et al. (2025) explicitly characterizing AI's role in architecture as "insufficiently understood."

RQ2 (Challenges): Three interlocking challenge dimensions were identified technical (quality, debt, integration), ethical/trust-related (bias, explainability, misinformation risk), and organizational (skills gap, governance, risk-averse delegation) which jointly explain the growing

divergence between AI usage rates and AI trust levels documented in the Stack Overflow (2024, 2025) surveys.

RQ3 (Future research directions): Building on Sections 4.1 and 4.2, five future research directions are proposed in Section 4.4.

4.4 Future Research Directions

1. Trustworthy and explainable AI4SE. Given the documented decline in developer trust despite rising usage (Stack Overflow, 2025), future research should prioritize explainability mechanisms specifically tailored to code- and architecture-level AI outputs, enabling developers to audit AI reasoning rather than accept opaque suggestions.
2. Empirical validation of AI in high-stakes SDLC phases. Because current empirical evidence concentrates heavily on coding and testing, future research should extend rigorous, controlled empirical evaluation to requirements engineering, architecture, and deployment/operations decision-making, where evidence remains comparatively thin (Bucaioni et al., 2025).
3. Human-AI collaboration models and skill evolution. As AI reshapes the software engineer's role from sole author to supervisor/validator of AI-generated artifacts, research is needed into effective human-AI pairing models, appropriate levels of automation, and the evolving skill sets required of software engineers (Hou et al., 2024).
4. Security and quality assurance for AI-generated code. Given evidence that a substantial share of committed code is now AI-generated (Getpanto, 2025) alongside persistent trust concerns, dedicated research into security vulnerability patterns, technical debt accumulation, and long-term maintainability of AI-generated code is needed.
5. Standardized benchmarks and quality models for AI-based software. Building on the finding that AI-based software quality differs meaningfully from traditional software quality, future research should develop and validate standardized, cross-study benchmarks and quality models specific to AI-integrated and AI-generated software artifacts.

4.5 Limitations of This Review

As a narrative rather than systematic review, this synthesis does not apply an exhaustive, replicable search protocol or formal risk-of-bias assessment to every primary study; instead, it draws primarily on existing high-quality systematic reviews and authoritative survey data. This approach maximizes breadth and integrative insight but may not capture every relevant recent primary study, particularly given the extremely rapid publication rate in the LLM4SE sub-field. Readers seeking exhaustive coverage of a specific SDLC phase are directed to the phase-specific systematic reviews cited throughout Section 3.

5. CONCLUSION

This narrative review synthesized current developments, challenges, and future research directions in AI-driven software engineering, drawing on systematic reviews, empirical studies, and industry survey data spanning 2019–2025. The evidence shows that AI adoption in software engineering has grown rapidly and unevenly across the SDLC, with code generation and testing considerably more mature than architecture and requirements engineering, and with high-stakes activities such as deployment and monitoring showing the lowest practitioner willingness to delegate to AI. A persistent and widening gap between rising AI usage and declining developer trust emerged as a central cross-cutting finding, explained through the integrative AI4SE Adoption and Impact Framework proposed in Section 4.2. Future research should prioritize explainability, empirical validation in under-researched high-stakes phases,

human-AI collaboration models, security and quality assurance for AI-generated code, and standardized quality benchmarks for AI-integrated software. As AI capabilities continue to advance, sustained empirical and conceptual research will be essential to ensure that software engineering practice evolves in a manner that is not only more productive, but also more trustworthy, secure, and human-centered.

6. REFERENCES

Ali, S. (2025). Enhancing software engineering with AI: Innovations, challenges, and future directions. IET Software, 2025, Article 5691460. https://doi.org/10.1049/sfw2/5691460

Bucaioni, A., Weyssow, M., He, J., Lyu, Y., & Lo, D. (2025). Artificial intelligence support for software architecture practice: A systematic review and future directions. ACM Transactions on Software Engineering and Methodology. https://doi.org/10.1145/3828546

DevOps.com. (2024, January 3). Measuring GitHub Copilot's impact on engineering productivity. https://devops.com/measuring-github-copilots-impact-on-engineering-productivity/

Garousi, V., Felderer, M., & Mäntylä, M. V. (2016). The need for multivocal literature reviews in software engineering: Complementing systematic literature reviews with grey literature. In Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (pp. 1–6). ACM. https://doi.org/10.1145/2915970.2916008

Getpanto. (2025). GitHub Copilot statistics 2026 Users, revenue & adoption. https://www.getpanto.ai/blog/github-copilot-statistics

GitHub. (2024, May 21). Research: Quantifying GitHub Copilot's impact on developer productivity and happiness. The GitHub Blog. https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). Large language models for software engineering: A systematic literature review. ACM Transactions on Software Engineering and Methodology, 33(8), Article 220. https://doi.org/10.1145/3695988

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering: A systematic literature review. Information and Software Technology, 51(1), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009

Kuwajima, H., & Ishikawa, F. (2019). Adapting SQuaRE for quality assessment of artificial intelligence systems. In 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) (pp. 13–18). IEEE. https://doi.org/10.1109/ISSREW.2019.00041

Lwakatare, L. E., Raj, A., Bosch, J., Olsson, H. H., & Crnkovic, I. (2019). A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In Agile Processes in Software Engineering and Extreme Programming (XP 2019) (pp. 227–243). Springer. https://doi.org/10.1007/978-3-030-19034-7_14

Nascimento, E., Alencar de Souza, B., Chagas Coelho, R., Nogueira de Araújo, A., Costa, L. F. S., & Souza, R. (2020). Software engineering for artificial intelligence and machine learning software: A systematic literature review. arXiv preprint. https://doi.org/10.48550/arXiv.2011.03751

Rêgo de Souza, T. R. et al. (2024). Generative artificial intelligence use in optimising software engineering process: A systematic literature review. Applied Computer Systems, 29(1). https://doi.org/10.2478/acss-2024-0009

Schäfer, M., Nadi, S., Eghbali, A., & Tip, F. (2023). An empirical evaluation of using large language models for automated unit test generation. IEEE Transactions on Software Engineering, 50(1), 85–105. https://doi.org/10.1109/TSE.2023.3334955

Souza, R. et al. (2021). Systematic literature review on software quality for AI-based software. Empirical Software Engineering, 26, Article 122. https://doi.org/10.1007/s10664-021-10105-2

Stack Overflow. (2024). 2024 Stack Overflow Developer Survey AI section. https://survey.stackoverflow.co/2024/ai

Stack Overflow. (2025). 2025 Stack Overflow Developer Survey AI section. https://survey.stackoverflow.co/2025/ai

Stack Overflow. (2025, December 29). Developers remain willing but reluctant to use AI: The 2025 Developer Survey results are here. Stack Overflow Blog. https://stackoverflow.blog/2025/12/29/developers-remain-willing-but-reluctant-to-use-ai-the-2025-developer-survey-results-are-here/

Yasin, A., Fatima, R., & Khan, M. (2025). A review of large language models for automated test case generation. Machine Learning and Knowledge Extraction, 7(3), Article 97. https://doi.org/10.3390/make7030097

Zafar, M. et al. (2025). Harnessing large language models for automated software testing: A leap towards scalable test case generation. Electronics, 14(7), Article 1463. https://doi.org/10.3390/electronics14071463