Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

Software Engineering Practices: In the era of AI / LLMs

Ashif Anwar
Independent Researcher, USA
Corresponding Author: Ashif Anwar, **E-mail**: reachaanwar@gmail.com

| ABSTRACT

The software development landscape is going through a major shift as generative artificial intelligence (AI) tools, including code assistants and autonomous agents, become increasingly widespread. Recent industry surveys indicate that more than 75% of developers currently use or plan to adopt AI-based solutions, with approximately half of professional developers utilizing these tools daily. Furthermore, organizational adoption is nearly universal, with reports suggesting that over 97% of companies incorporate AI into their development workflows for tasks such as code generation, documentation, code review, and automated testing. The anticipated benefits of these technologies are considerable. Empirical studies report productivity gains of up to 50% in coding speed and 33% in code refactoring efficiency. However, these advantages are accompanied by notable risks. AI-generated code can propagate security vulnerabilities and exacerbate technical debt. Moreover, evidence from a 2025 randomized controlled trial (RCT) shows that experienced developers using AI assistants were, on average, 19% slower when completing real-world programming tasks, challenging the assumption of universal efficiency gains. This paper discusses the current state of generative AI adoption in software engineering and synthesizes best practices for employing its potential while controlling associated risks. Following the structure recommended by the Journal of Computer Science and Technology Studies, the discussion is organized into the following sections: Introduction, Literature Review, Methodology, Results and Findings, Conclusion, Statements and Declarations, and References [11].

| KEYWORDS

Generative AI; Software Development; AI Augmented SDLC; Best Practices; Developer Productivity; AI Ethics

| ARTICLE INFORMATION

ACCEPTED: 20 December 2025 **PUBLISHED:** 29 December 2025 **DOI:** 10.32996/jcsts.2025.7.12.57

1. Introduction

Generative AI models built on large language models (LLMs) have demonstrated the ability to produce coherent text, executable code, and even system-level designs. Since the introduction of tools like GitHub Copilot and ChatGPT, software teams have progressively embraced AI to accelerate their development workflows. By late 2023, surveys reported that approximately 75% of developers were using AI-powered code assistants, with LLM-based tools reducing routine programming time by nearly 50% [3]. Novel paradigms such as chat-oriented programming (CHOP), vibe coding, and agentic programming have emerged, permitting developers to interact with conversational agents rather than writing code manually [3].

Adoption intensified in 2024 and 2025. A Techreviewer survey in 2025 revealed that 97.5% of organizations integrated AI into at least one stage of the software development life cycle (SDLC) [2]. Similarly, the Stack Overflow Developer Survey 2025 found that 84% of respondents were using or planning to use AI tools, with 50.6% of professional developers employing them daily [1]. Today, AI permeates the SDLC from natural language requirements analysis to automated test generation and deployment scripting. Nevertheless, advances and still unresolved obstacles continue. AI-generated code can appear correct yet contain subtle errors, replicate outdated or insecure patterns [4], and encourage copy-paste practices that increase technical debt [5]. Operational outlays are also rising; the computational expense of large models grew by 89% between 2023 and 2025 [3]. Furthermore, issues of intellectual property, privacy, and accountability are still controversial [7]. Empirical evidence suggests productivity gains are not universal: a randomized controlled trial in early 2025 found that experienced open-source developers using AI assistants were 19% slower on real-world tasks [6].

To achieve sustainable benefits, organizations must integrate AI within established engineering practices such as version control, test-driven development, code review, and continuous integration/deployment [8]. This study combines findings from academic literature, industry surveys, and reports published between 2023 and late 2025. We identify the areas where AI has the greatest impact, highlight associated risks and trade-offs, and propose best practices for responsible adoption. Additionally, we present original visualizations illustrating AI adoption trends and its augmentation of traditional development phases.

2. Literature Review

2.1 AI Adoption and Integration Across the SDLC

Early studies documented the rapid uptake of AI coding assistants. Generative AI and the Transformation of Software Development Practices reported that by mid-2023, approximately 75% of developers were using AI tools, attaining up to 50% faster coding and 33% faster refactoring, while introducing paradigms such as chat-oriented programming and agentic programming [3]. The DFKI/Accenture study additionally highlighted that AI influences every phase of the software development life cycle (SDLC). Activities traditionally performed by humans, such as requirements extraction, design, and testing, can now be partially automated, authorizing programmers to concentrate on higher-level tasks, including problem formulation, orchestration, and supervision [7].

Industry surveys reinforce this evidence. The Techreviewer 2025 survey revealed that 97.5% of companies have integrated AI into their development workflows, leaving only 2.5% yet to adopt [2]. Adoption is particularly strong in code generation (72.2%), documentation (67.1%), code review (67.1%), automated testing/debugging (55.7%), and requirements analysis/design (53.2%) [2]. Notably, the same report points out that AI use has expanded upstream, with more than half of companies now employing AI for requirements analysis and design, activities that require context-sensitive comprehension [2]. Similarly, the Stack Overflow Developer Survey 2024 indicated that 76% of developers used or planned to use AI tools, with 62% already using them [9]. By 2025, these figures rose further, as 84% of respondents reported using or planning to use AI, and 51% of professional developers used AI tools daily [1].

2.2 Benefits and Emerging Paradigms

Developers embrace AI because it accelerates mundane tasks and supports creative workflows. GitHub's internal research, cited in the Stack Hawk blog, reports that AI assistants enable engineers to complete routine tasks 55% faster [4]. Early studies noted improved work rate and boosted developer satisfaction. New paradigms such as chat-oriented programming, vibe coding, and agentic programming encourage developers to articulate intent in natural language and rely on AI agents to generate or modify code [3]. High-performing organizations described by McKinsey embed AI across the entire product development life cycle and develop AI-native roles (e.g., prompt engineers, AI product owners) to orchestrate multi-agent systems [10]. These companies measure impact carefully and provide upskilling programs to ensure their workforce can effectively use AI [10].

2.3 Risks and Challenges

Despite large-scale adoption, AI brings major risks. Security vulnerabilities are a central concern: AI models sometimes generate insecure code, import vulnerable dependencies, or replicate anti-patterns [4]. The StackHawk report warns that traditional security reviews cannot keep pace with the speed at which AI produces code [4]. Furthermore, AI tools may increase code churn and technical debt by encouraging copy-and-paste behaviors [5]. Credibility and liability are unresolved, as generative models occasionally hallucinate functionality or produce legally uncertain outputs, and the cost of training and operating large models rose by nearly 89% between 2023 and 2025 [3].

Empirical evidence of productivity gains is also mixed. While many developers perceive AI as a driver of speed, a 2025 randomized controlled trial (RCT) involving 16 experienced open-source developers found that allowing AI assistance slowed developers by 19% on real issues [6]. Participants anticipated a 24% speedup but still believed they had been faster, even when they were slower, highlighting a gap between perception and reality [6]. This study shows that AI can introduce overhead when developers need to validate, debug, or rework AI-generated suggestions. It highlights the need for thorough evaluation and integration with established engineering processes.

2.4 Best Practices in Traditional Software Engineering

High-quality software development relies on structured workflows. Version control systems such as Git, along with structured branching and merging strategies, enable teams to track changes, collaborate, and roll back when necessary [8]. Test-driven development (TDD) advocates writing failing tests before implementing functionality and refactoring after tests pass, guaranteeing that code meets requirements and remains maintainable [8]. Code reviews and pair programming promote knowledge sharing and catch defects early [8]. Continuous Integration/Continuous Deployment (CI/CD) automates building, testing, and deployment, enabling frequent releases whilst upholding quality [8]. Agile and iterative development emphasize small, incremental improvements and close partnership with stakeholders [8]. Clean code and refactoring practices focus on readability and maintainability [8], and automated testing strategies, including unit, integration, and system tests, catch defects early and support continuous delivery [8].

3. Methodology

This paper adopts a multi-faceted approach strategy that combines a systematic literature review with descriptive analyses of recent surveys and reports. academic repositories such as arXiv and the ACM Digital Library, as well as industry sources including Techreviewer, Stack Overflow, McKinsey, DFKI/Accenture, StackHawk, DevOps.com, and METR, were searched for publications published between January 2023 and December 2025. Search terms included “generative AI,” “software engineering,” “code assistant,” “AI adoption,” “productivity,” “security,” and “best practices.” Sources were selected if they provided quantitative data on AI adoption or qualitative observations regarding its impact on software development. In total, over twenty publications were reviewed, but only those relevant to adoption, productivity, risks, and best practices were synthesized.

Original diagrams were created to visualize AI augmentation of the SDLC and adoption trends across use cases. Data for the adoption chart were extracted from the Techreviewer 2025 survey and the Stack Overflow Developer Survey 2024–2025 [2][1]. The diagrams were generated using Python’s Matplotlib library and are included as figures in the Results section.

4. Results/Findings

4.1 AI-Augmented Software Development Life Cycle

The classical SDLC comprises six phases: requirements analysis, design and architecture, implementation, testing, deployment, and maintenance. Figure 1 illustrates how AI augments each phase. During requirements analysis, natural language processing (NLP) models extract user stories, generate acceptance criteria, and detect ambiguities. In design, generative models put forward architectures, design patterns, and user interface layouts. Implementation is the most widely adopted phase, where AI code assistants generate boilerplate code, suggest completions, and refactor functions. In testing, AI generates unit tests, identifies edge cases, and performs fuzzing. Deployment benefits from AI-enabled infrastructure-as-code scripts, anomaly detection, and automated rollback strategies, while maintenance leverages AI for log analysis, automated bug triage, and self-healing. Nonetheless, benefits, human monitoring remains vital to validate AI outputs, align solutions with corporate aims, and ensure ethical and secure outcomes.

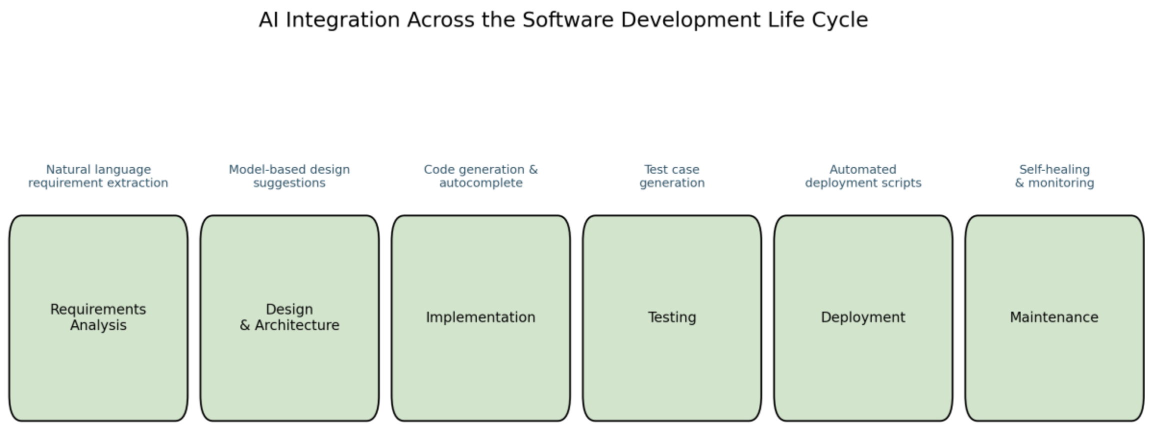
AI Integration Across the Software Development Life Cycle
Natural language requirement extraction
Model-based design suggestions
Code generation & autocomplete
Test case generation
Automated deployment scripts
Self-healing & monitoring
Requirements Analysis
Design & Architecture
Implementation
Testing
Deployment
Maintenance

Fig 1: AI integration across the software development life cycle (SDLC).

4.2 AI Adoption Across Use Cases (2024 vs 2025)

Figure 2 compares adoption rates for major AI use cases between 2024 and 2025. Data from the Techreviewer survey shows that code generation remains the most common use case, adopted by about 72% of companies in 2025 [2]. Documentation generation and code review/optimization saw significant increases, indicating the shift toward AI-assisted knowledge management. Automated testing/debugging and requirements analysis/design also grew, signaling that AI adoption is moving upstream in the

SDLC. A small "Other" category includes tasks such as DevOps automation, UI/UX optimization, and predictive analytics [2]. When compared with the Stack Overflow survey, which showed that 84% of developers are using or planning to use AI tools [1], the figures suggest that organizations are integrating AI comprehensively and that individual developers are increasingly comfortable relying on AI for various tasks.

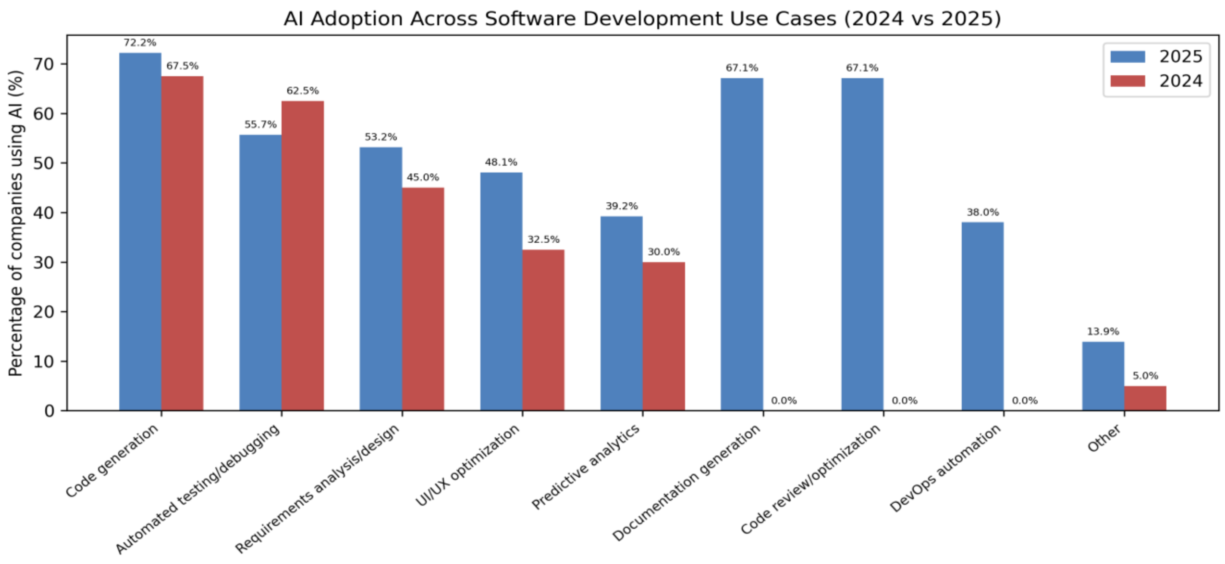

Fig 2: AI adoption across software engineering use cases in 2024 and 2025.

4.3 Productivity Effects and Developer Experience

Survey respondents generally report positive productivity impacts from AI, though experiences vary. In GitHub's study, developers performed routine tasks 55% faster with AI assistants [4]. The generative AI paper observed 50% faster coding and 33% faster refactoring [3]. Conversely, the METR RCT found that experienced developers were 19% slower when allowed to use AI tools [6]. This slowdown might be explained by time spent interpreting AI suggestions, debugging hallucinated code, and aligning AI output with project settings [6]. The discrepancy between benchmark results and actual usage underscores the need for more nuanced assessment criteria.

AI also affects developer morale and satisfaction. Some developers appreciate AI's ability to handle mundane tasks, leaving them to focus on higher-level design and problem-solving. Others show irritation when AI suggestions require significant rewriting or introduce subtle bugs. The Stack Overflow 2025 survey reported that, despite high adoption, favorable sentiment toward AI decreased from over 70% in 2023–2024 to about 60% in 2025, suggesting that expectations are being recalibrated [1].

4.4 Security, Quality and Best Practices

AI-generated code may omit secure code development standards, include vulnerable dependencies, or replicate anti-patterns [4]. To handle these risks, security researchers recommend configuring AI tools with security rules, performing automated security testing, and integrating security scanning into CI/CD pipelines [4]. DevOps practitioners caution that AI code assistants can increase code churn and accelerate the accumulation of technical debt, so teams should set clear quality guidelines, adopt strong automated testing, establish feedback loops to improve prompts, and evaluate AI output utilizing metrics beyond lines of code [5]. Education is fundamental: developers must understand AI's limitations, avoid excessive dependence on generated code, and refine prompts to obtain better results.

Integrating AI into traditional practices requires adaptation. Version control and structured Git workflows remain fundamental for managing AI-generated code and enabling collaborative review [8]. Test-driven development should be extended to verify AI-generated code against pre-written tests [8]. Code reviews and pair programming are essential for validating AI suggestions and for sharing knowledge [8]. Continuous Integration and Deployment should automate the testing of AI-produced artifacts and support rapid iteration [8]. Agile methods provide the leeway to innovate with AI tools and adjust processes based on feedback [8]. These practices create guardrails that help teams harness AI's speed whilst preserving quality and security.

5. Conclusion

Generative AI has transitioned from novelty to mainstream adoption in software engineering. Surveys show overwhelming use of AI tools, with nearly every organization integrating them into at least one development phase and more than half of professional developers using AI daily [2][1]. AI assists across the SDLC, from requirements analysis to maintenance, and accelerates tasks such as code generation, testing, and documentation. The gains are considerable: productivity gains of up to 50%, new programming paradigms, and avenues for creativity. However, these advantages come with caveats. Empirical evidence reveals that AI does not always accelerate development—an early 2025 RCT found a 19% slowdown [6]—and AI-generated code may contain vulnerabilities [4] or increase technical debt [5]. The field also faces unresolved ethical and judicial questions and escalating computing costs [3].

To handle this era, software teams must integrate AI within established engineering frameworks. Version control, test-driven development, code review, continuous integration and deployment, agile methods, refactoring, and automated testing provide guardrails that ensure AI-augmented development persists as secure, reliable, and maintainable [8]. Organizations should train developers to use AI responsibly, configure tools with security rules, and measure the impact of AI adoption on quality and speed [10][5]. Researchers should conduct further studies to understand long-term productivity effects and develop benchmarks that represent real-world developer workflows. By combining AI's strengths with disciplined engineering practices, the software community can utilize generative AI to build better, safer, and more innovative systems.

Statements and Declarations

Funding: This research received no external funding.

Conflicts of Interest: This author declares no conflict of interest.

ORCID ID: 0009-0002-1009-3864

Acknowledgements: The author thanks the creators of open-source data and surveys used in this work.

References

[1] Stack Overflow. (2025). 2025 developer survey: AI section. Stack Overflow.
https://survey.stackoverflow.co/2025/ai
[2] Techreviewer. (2025). AI in software development 2025: From exploration to accountability.
A global survey analysis. *Techreviewer*. https://techreviewer.co/blog/ai-in-software-development-2025-from-exploration-to-accountability-a-global-survey-analysis
[3] Acharya, V. (2025). Generative AI and the transformation of software development practices (arXiv No. 2510.10819). arXiv. https://doi.org/10.48550/arXiv.2510.10819
[4] StackHawk. (2025). Secure AI development: Balancing speed & code security. StackHawk Blog. https://www.stackhawk.com/blog/secure-ai-software-development/
[5] DevOps.com. (2025). Best of 2025: AI in software development: Productivity at the cost of code quality? DevOps.com. https://devops.com/ai-in-software-development-productivity-at-the-cost-of-code-quality-2/
[6] Becker, O., Singer, J., & MacEwen, J. (2025). Measuring the impact of early-2025 AI on experienced open-source developer productivity: A randomized controlled trial. *METR*. https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
[7] Accenture & German Research Center for Artificial Intelligence (DFKI). (2025). Generative AI in software engineering: Transforming the software development process. Accenture & DFKI. https://www.dfki.de/fileadmin/user_upload/DFKI/Medien/News/2025/Wissenschaftliche_Exzellenz/Generative_AI_in_Software_Engineering_Transforming_the_Software_Development_Process_2025.pdf
[8] Zest. (2025). Top software engineering best practices for 2025. *Zest Blog*. https://meetzest.com/blog/software-engineering-best-practices
[9] Stack Overflow. (2024). 2024 developer survey: AI section. Stack Overflow. https://survey.stackoverflow.co/2024/ai
[10] McKinsey & Company. (2025). The state of AI in 2025: Agents, innovation, and transformation.
https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai
[11] Lamm, D. V., & Reed, T. (2010). Demographics of the Contracting Workforce within the Army Contracting Command. https://core.ac.uk/download/36726431.pdf