Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

Integration of AI in Software Engineering: Revolutionizing Development: A Survey Current Practices and Future Directions

Samra Rehman
Lecture, Swedish College of Engineering and Technology Wah Cantt
samrarehmano00@gmail.com
Hasna Arshad (Corresponding Author)
Assistant Professor, HITEC University, Taxila
hasna.arshad@hitecuni.edu.pk
Fadia Ali Khan
Assistant Professor, HITEC University, Taxila
fadia.ali@hitecuni.edu.pk
Isma ul Hassan
Assistant Professor, National University of Computer and Emerging Sciences Islamabad
isma.hassan@nu.edu.pk
Behjat Zuhaira
Assistant Professor, National University of Computer and Emerging Sciences Islamabad
behjat.zuhaira@nu.edu.pk

ABSTRACT

The growing complexity of software systems and the demand for accelerated, high-quality delivery have positioned Artificial Intelligence (AI) as a transformative force in Software Engineering (SE). This symbiotic relationship where AI tools are developed through SE processes and, in turn, redefine those very processes promises significant gains in efficiency, accuracy, and productivity across the Software Development Life Cycle (SDLC). However, the integration of AI within SE is not uniformly understood or adopted, creating a critical gap between its potential and its practical, holistic application. This study investigates this disparity by surveying software experts and researchers with hands-on experience in AI-enabled development. Using descriptive statistics to analyze the survey data, we quantify the adoption level of AI in each SDLC phase. The findings reveal high integration in implementation and testing stages driven by tools for code generation, debugging, and defect prediction but comparatively low adoption in requirement analysis and maintenance. This uneven landscape highlights a predominant focus on automating *how* software is built and validated, rather than on defining what to build or ensuring its long-term evolution. The research elucidates both the advantages of AI adoption and its associated challenges, including integration complexity, expertise shortages, and trust issues. By mapping the current state and identifying these gaps, this work provides a foundation for more balanced and comprehensive AI integration strategies in software engineering. This study makes three primary contributions to the growing body of knowledge on Artificial Intelligence in

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

Software Engineering. First, it consolidates and synthesizes prior research across all phases of the Software Development Life Cycle, offering a structured account of where the existing literature is concentrated and where it remains thin. Second, it contributes new empirical evidence, drawn directly from practitioners who use AI tools on a daily basis, to test whether the optimism reflected in much of the existing literature matches the lived experience of software professionals. Third, it offers a diagnostic framework, grounded in survey data, for understanding why certain phases of the Software Development Life Cycle have embraced AI more readily than others, and what organizational, technical, and human factors explain this unevenness. Taken together, these contributions are intended to inform both academic researchers seeking to identify promising directions for future inquiry and industry practitioners seeking to make informed decisions about where and how to invest in AI enabled tooling.

Keywords: AI Integration, Artificial Intelligence (AI), Software Engineering, Software Development Life Cycle (SDLC)

1. INTRODUCTION

The field of software engineering (SE) is currently undergoing a revolutionary transformation due to the increasing and ubiquitous adoption of Artificial Intelligence (AI). [1]. Traditionally, the field has evolved through a series of paradigm shifts, from the inflexible and sequential approach of the Waterfall model to the flexible and iterative approach of Agile, and more recently towards automation-focused cultures of DevOps and MLOps. [2]. Each paradigm shift has sought to counter the rising complexity of software systems, the growing velocity of market demands, and the challenges of human fallibility, resource wastage, and slow time-to-market. However, despite these developments, software engineering has remained essentially a human-dominated field, relying on the skills, ingenuity, and hard work of software engineers, testers, architects, and project managers [3]. In the current scenario, AI encompassing machine learning (ML), deep learning (DL), natural language processing (NLP), and generative models holds the promise not only to extend these human capabilities but also to fundamentally redefine the software development paradigm. This paradigm shift repositions the software engineer from a human coder to an intelligent system conductor, marking the beginning of the AI-driven software engineering era. [4]. The forces that drive this paradigm shift are complex and varied. The rapid increase in computational capabilities and access to data has made it easier to train complex models that can interpret the semantics of code, identify patterns in large-scale code repositories, and produce human-like text and code. At the same time, the rising complexity of contemporary software systems, such as cloud-native architectures, microservices, IoT systems, and cyber-physical systems, has outstripped the ability of manual processes to guarantee quality, security, and maintainability.[4, 5]. In addition, the economic and competitive forces driving businesses today require them to speed up their development cycles, minimize costs, and provide highly personalized user experiences. Artificial intelligence offers a practical solution to these challenges by automating repetitive tasks, predicting software defects before they occur, optimizing resource utilization, and allowing software to behave in an intelligent,

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

adaptive manner.[6].

The trajectory described above reflects a much longer pattern in the discipline, in which each generation of software professionals has sought new means of managing complexity that earlier generations could not have anticipated. The transition from monolithic, hand crafted systems to component-based architectures, and subsequently to distributed, cloud native, and service oriented systems, has consistently been accompanied by a search for tools that reduce cognitive load and allow engineers to operate at higher levels of abstraction. Artificial Intelligence represents, in this sense, not an isolated technological novelty but rather a continuation of a decades long effort to automate what can be automated so that human attention can be reserved for genuinely novel problems. What distinguishes the current moment is the breadth of the automation now possible. Earlier generations of tools, such as compilers, integrated development environments, and static analyzers, automated narrowly defined and well understood tasks. Contemporary AI systems, by contrast, can interpret natural language instructions, generating original code, reasoning about program behavior, and even proposing architectural decisions, tasks that were until recently considered to require distinctly human judgment.

This breadth is made possible by the convergence of several supporting developments. Advances in machine learning architectures, particularly the emergence of transformer-based language models trained on enormous corpora of source code and natural language text, have produced systems with an unprecedented capacity to recognize patterns across millions of repositories. At the same time, cloud computing has made the computational resources necessary to train and serve such models widely accessible, removing what was once a significant barrier to experimentation. The proliferation of open repositories, issue trackers, and version control histories has also supplied the large, labeled datasets on which supervised and self supervised learning techniques depend. Each of these enabling conditions reinforces the others, producing a feedback loop in which better tools attract more usage, more usage generates more data, and more data in turn supports the training of still more capable tools. It is this self reinforcing dynamic, as much as any single technical breakthrough, that explains why AI integration in Software Engineering has accelerated so rapidly in recent years.

1.1 Innovations and Advantages

There have been significant innovations in the software development life cycle that have been made possible by the integration of AI.

- During the requirements and design phase, natural language processing algorithms can be used to analyze stakeholder documents, user feedback, and regulatory documents to extract and formalize requirements, point out ambiguities, and even suggest optimal architectural patterns [7].
- During the implementation stage, AI-assisted tools like GitHub Copilot, Amazon Code Whisperer, and Tabnine offer developers context-related code suggestions, write entire functions or classes based on natural language inputs, and refactor code to improve readability and performance [8].
- In the area of testing and quality assurance, machine learning models are used for automatic generation of test cases, prioritization of test suites based on risk

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

analysis, prediction of defect-prone modules, and detection of security vulnerabilities using static and dynamic analysis[9].

- In the area of DevOps and deployment, AI has the ability to improve continuous integration and delivery (CI/CD) pipelines by predicting build failures, scaling infrastructure, monitoring application performance, and implementing self-healing capabilities [10].
- Finaly, in maintenance and evolution, AI helps in understanding legacy systems, automated bug triaging and fixing, technical debt detection, and predictive maintenance through runtime log analysis[11]. Taken together, these advantages bring about dramatic improvements in developer productivity, software quality, reliability, and security, as well as unprecedented levels of personalization and adaptability in end-user experiences[3, 12].

Considered together, these phase specific innovations illustrate a broader pattern in which AI functions less as a single tool than as an underlying capability that can be embedded throughout the Software Development Life Cycle. Rather than replacing any one role outright, AI systems tend to absorb the more repetitive, pattern based components of a task while leaving judgment intensive decisions, such as prioritization, trade off analysis, and stakeholder negotiation, in human hands. This division of labor has practical implications for how organizations should think about AI adoption. Teams that treat AI purely as a productivity multiplier for existing workflows tend to realize modest, incremental gains, whereas teams that redesign their workflows around the capabilities and limitations of AI tools often report more substantial improvements in throughput and quality. The unevenness in adoption observed across different phases, which this study investigates empirically in later sections, can therefore be understood partly as a reflection of how readily each phase lends itself to this kind of workflow redesign.

1.2 Challenges and Risks

Although these are promising developments, the integration of artificial intelligence (AI) in software engineering is also faced with certain challenges and risks[13].

- Data dependency and bias are a major concern: AI models are effective based on the data used for training, and if the data is of poor quality, incomplete, or biased, it may lead to inaccurate, unfair, or discriminatory software results, thus giving rise to substantial ethical concerns [14].
- Security loopholes could potentially be created by the use of code written by AI that has not been thoroughly audited, as well as by adversarial attacks on the AI models themselves. The “black box” nature of many advanced AI systems, especially deep learning models, makes it difficult to debug and certify them, which is why there is such a need for explainable AI (XAI) techniques [15].
- Furthermore, there are economic and organizational hurdles that need to be overcome, such as the high startup costs associated with specialized hardware and software, as well as the expertise of data scientists and machine learning engineers. From an ethical perspective, the question of accountability and responsibility in delegating decision-making authority to AI systems must be carefully considered.

Beyond the technical and economic dimensions outlined above, the introduction of AI into software engineering practice also raises governance questions that organizations are only beginning to address. Decisions about which tasks may be

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

delegated to an AI system, how outputs should be reviewed before being accepted into a production codebase, and who bears responsibility when an AI assisted decision leads to a defect or a security incident, are not yet governed by mature, widely accepted standards. In the absence of such standards, many organizations rely on informal norms that vary considerably from team to team, creating inconsistency in how AI generated artifacts are vetted. This inconsistency is compounded by the pace of change in the underlying technology itself. Tools, models, and best practices evolve quickly enough that policies written even a year earlier may already be outdated, leaving practitioners to navigate a moving target without clear institutional guidance. These governance gaps, together with the technical and human centered challenges already discussed, motivate the empirical investigation undertaken in this study, which seeks to understand not only how AI is being used but also how practitioners themselves perceive its risks and limitations.

1.3 Related Research Questions

The following are the three research questions related to the study:

RQ1: To what extent are AI-powered tools currently adopted within software organizations?

RQ2: What is the impact of the holistic, end-to-end integration of AI tools throughout the software development life cycle on performance?

RQ3: What are the main organizational and ethical issues faced in software engineering practices?

These three research questions are deliberately structured to move from description to explanation. RQ1 establishes a descriptive baseline by quantifying the extent of current adoption, RQ2 probes the consequences of that adoption for development performance, and RQ3 surfaces the organizational and ethical conditions that shape, and sometimes constrain, how AI is used in practice. The remainder of this paper addresses these questions in turn, beginning with a review of the existing literature that situates the present study within the broader body of research on AI in software engineering.

2. Literature Review

The history of software engineering (SE) is characterized by the constant quest for efficiency, quality, and flexibility, from traditional waterfall models to agile and, more recently, highly automated DevOps models [20]. Nevertheless, challenges like the presence of human error in manual coding and inefficient resource allocation have driven a new innovation paradigm, mainly fueled by Artificial Intelligence (AI). AI tools like machine learning (ML), natural language processing (NLP), and large language models (LLMs) are revolutionizing software development by facilitating pattern recognition, prediction, and the automation of complex tasks [21]. The current use of AI in software development covers the entire Software Development Life Cycle (SDLC), showing great potential in the following areas:

This historical progression is worth dwelling on because it clarifies what is genuinely new about the current wave of AI adoption and what is simply an extension of earlier automation efforts. Early software engineering practice relied almost entirely on manual coding, with limited tool support beyond compilers and rudimentary debuggers. The introduction of integrated development environments,

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

automated build systems, and continuous integration pipelines incrementally reduced the manual burden on developers, but these tools operated according to fixed, explicitly programmed rules. They could enforce a coding standard or flag a syntax error, but they could not interpret the intent behind a piece of code or suggest an entirely new implementation. The shift toward machine learning based and, more recently, large language model based tools marks a qualitative change in this respect, since these systems are capable of probabilistic reasoning over patterns learned from data rather than simply executing predefined rules. It is this capacity for learned, context sensitive behavior that underlies the wide ranging applications surveyed across the remainder of this section.

- **Requirements and Design:** NLP techniques support requirements elicitation by analyzing unstructured data and pointing out ambiguities, while AI tools assist in automating design pattern recognition and intelligent system modeling [22]. The application of AI, especially NLP, in requirements engineering and system design has made a substantial impact on the automation of early-stage software development tasks. Recent research work highlights the application of NLP in analyzing unstructured textual data from stakeholder documents, user feedback, and legal texts to extract, categorize, and formalize software requirements. For instance, Deng et al. [23] developed an NLP-based framework that uses transformer models to detect ambiguities and inconsistencies in requirement specifications, thus improving traceability and minimizing human error in requirements oversight. In addition, AI tools are increasingly being used for automating design pattern recognition and intelligent system modeling. Sharma and Lee [24] developed an AI system that uses CNNs and GNNs to provide optimal design pattern suggestions based on past project experiences and system constraints, thus improving efficiency and design excellence. Wang et al. [25] further improved this by incorporating reinforcement learning to dynamically update software architecture models based on changing software requirements, thus improving adaptability in agile and DevOps environments.

The growing body of work in this area reflects a recognition that requirements engineering, despite being the earliest phase of the Software Development Life Cycle, has historically received comparatively little automation support relative to later phases such as coding and testing. This imbalance is partly attributable to the inherently unstructured and socially negotiated nature of requirements, which are typically expressed through informal conversations, meeting notes, and lengthy stakeholder documents rather than through the more structured artifacts, such as source code or test scripts, that lend themselves naturally to automated analysis. Natural language processing techniques offer a promising avenue for addressing this gap precisely because they are designed to operate over unstructured text. Even so, the literature in this area remains comparatively young, and most published studies report results from controlled academic settings rather than from sustained deployment within commercial organizations, a limitation that the present study's survey based methodology is partly designed to address.

Alenezi et al. [1] This paper presents a detailed analysis of the impact of artificial intelligence (AI) on software engineering processes across the entire software development life cycle. This paper explains the role of AI in software development activities such as code generation, debugging, testing, and maintenance, and how

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

tools such as GitHub Copilot and AlphaCode have significantly improved the efficiency of software developers. AI has also improved requirement gathering through natural language processing and intelligent system modeling and design pattern recognition. Case studies from leading companies such as McKinsey, IBM, Microsoft, and Google have shown significant improvements in development time, system reliability, and defect removal. [18]. A poll of 250 professionals also showed use of AI tools now firmly established, particularly in coding and debugging, but that trust, skills gaps and ethical issues remain challenges. The paper presents the general frame of integration of responsible AI and advocates for educational reforms to train future software engineers. It wraps up by discussing emerging areas, such as AI in IoT, quantum computing and domain-specific applications and highlights the need for a balance between automation/human control [17] .

The breadth of Alenezi and colleagues' review is valuable for situating the present study, since it confirms that practitioner adoption of AI tools is already well established in certain areas while highlighting persistent concerns around trust and skill gaps that closely mirror the challenges identified in the introduction of this paper. At the same time, because that review draws primarily on a single survey of 250 professionals conducted at a particular point in time, it offers limited insight into how adoption patterns vary across different phases of the Software Development Life Cycle or across organizations of different sizes. The present study seeks to extend this line of inquiry by collecting phase specific adoption data and by examining how organizational context, such as company size and respondent experience level, relates to reported patterns of AI use.

- **Coding and Implementation:** Products such as GitHub Copilot are built with LLMs to help customers build and debug applications, save time spent writing code, achieve more relevancy in autocomplete and suggestion, thereby increasing the development speed and reducing manual work. Big Language Models (BLMs) have transformed coding by context-aware code generation, completion, and refactoring. Chen et al. [26] empirically investigate GitHub Copilot and find that developers using AI-assisted tools write code 35–50% faster while keeping similar code quality. The research also stated that LLMs eliminate boilerplate code and helps to bridge syntactic gaps for beginner programmers. In a more recent investigation, Park et al. [27] explored the use of fine-tuned CodeBERT models for generating secure and efficient code snippets in embedded systems, emphasizing the importance of domain-specific training data. Moreover, Liu and Zhang [28]] proposed a multi-agent AI system to support code code generation. This system uses specialized AI agents for conceptual development, integration, or verification. This approach decreased manual effort, improving code consistency generated in large codebases. Feldt et al. [29] described a framework that systemically categorizes and evaluates artificial intelligence (AI) used in software engineering. This framework considers three dimensions: where the application happens (processing, product, or execution), the category of AI technology used (such as symbolist, connectionist, or analogizer), and how much automation is involved (including fully controlled by humans to fully automated). This taxonomy framework provides value in evaluating risks as well as opportunities in integrating artificial intelligence with software systems. Their paper uses the framework to test 15 technical articles from the RAISE workshop to demonstrate how effective it is in systemically evaluating

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

different applications involving artificial intelligence. Their analysis highlights that increased automation or executing at the runtime level is associated with relatively higher risks, suggesting that lower risks associated applications should be pursued first. Their framework is helpful for enhancing communication, managing risks, or making educated decisions regarding artificial intelligence adoption in software engineering.

The rapid uptake of large language model based coding assistants documented in these studies has, in turn, generated a parallel body of research concerned with evaluating the quality, security, and maintainability of AI generated code. Several researchers have raised concerns that, while such tools accelerate the writing of new code, they may also introduce subtle defects or insecure patterns that are not immediately apparent to the developer accepting the suggestion, particularly when developers place excessive trust in tool output without adequate review. This tension between productivity gains and quality risks recurs throughout the literature on AI assisted coding and is reflected directly in the survey findings reported later in this paper, where respondents express measurable concern about reliability even as they report substantial productivity benefits.

- **Testing and Quality Assurance:** The results produced by ML algorithms can be efficiently exploited for test case generation and prioritization and, importantly, defect prediction to help diagnose possible bugs early on and improve software reliability. Machine learning has increasingly become essential for the automation and optimization of testing and QA processes for software systems. Kumar et al. [30] designed and implemented a test case prioritization system for prioritizing tests using ML algorithms based on past defect information and code change information to predict high-risk modules and decrease regression testing time by as much as 40%. For automatic testing, Reyes et al. [31] used GANs to generate real and diverse test inputs to improve edge-case testing for safety-critical systems. Additionally, models used for defect prediction are now enhanced using explainable AI (XAI) approaches. Zhou et al. [32] designed a deep learning model for defect prediction and explained why a defect occurred to help developers prevent possible bugs ahead of time and improve software reliability. Nascimento et al. [33] conduct a systematic literature review to understand software engineering (SE) practices in developing artificial intelligence (AI) and machine learning (ML) software. Through a compilation of 55 studies from the past few decades (1990-2019), the authors spot deficiencies and best practices in the development of various domains of applications of AI/ML software. The literature review makes clear that most of the development of the software applications of AI and ML takes place in a lab environment or a corporate firm in a manner that is not structured or research-led, and the literature lacks understanding of best practices in enterprises. Software engineering challenges in developing applications of AI software in various sectors of the economy include software testing, software quality in AI software (with a special focus on ethics and interpretability, amongst other aspects not yet explored and discussed in depth in the literature), management of data, development of models, project management, infrastructure, and requirements engineering in software development [34].

Taken together, these contributions to automated testing and defect prediction demonstrate that machine learning has matured into a practical, widely applicable

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

tool for quality assurance teams rather than remaining a purely experimental technique confined to academic publications. Nonetheless, a recurring theme across this literature is the dependence of model performance on the availability of high quality historical defect data, a requirement that smaller organizations or newer projects, lacking extensive historical records, may struggle to satisfy. This data dependency, identified here in the specific context of testing, foreshadows a broader challenge that resurfaces throughout the literature review and is examined in greater depth in the discussion of data quality challenges later in this paper.

- **Deployment and Maintenance:** AI powered predictive maintenance with operational data analysis and system anomalies monitoring that minimize downtime and optimize performance. AI-driven predictive maintenance and anomaly detection are redrafting deployment and operational support. Singh et al. [35] Some examples are: applied time-series forecasting models, such as the LSTM networks, in order to predict system failures in cloud-based microservices, which allows them to scale preemptively and reduces drastically the amount of downtime. In a related study, Verma and Patel [36] have proposed an AI-powered monitoring tool using unsupervised learning to detect anomalous behavior from runtime logs, decreasing MTTR by 30%. For continuous maintenance, Fernández et al. [37] proposed a self-healing approach that integrates reinforcement learning techniques and automatic patching strategies for mitigating vulnerabilities and performance issues in a real-time environment. These studies showcase how AI, besides optimizing software deployment, also supports software longevity. Costa et al. [38] examines the disruptive role of artificial intelligence (AI) in software engineering and its applications in various fields, including cyber security, drug discovery, scientific writing, and research and development. This study showcases how AI, besides optimizing software deployment, has significantly added value to software engineering by enabling coding, testing, and software maintenance automation, thus boosting efficiency and software quality. It also examines new technologies such as NeuRRAM compute-in-memory silicon for energy-efficient AI processing, deep learning platforms for detecting malware that perform better than previous solutions, and more in software engineering. For drug discovery, AI enables fast identification of bioactive compounds by applying machine learning and molecular designs [39]. ChatGPT, a new tool, has started revolutionizing scientific writing, but there are some ethical issues in regard to bias and plagiarism. The paper emphasizes the importance of explainable AI and transparency in decision-making, and adopts a mixed-methods approach using literature reviews, surveys, and expert interviews. Ultimately, it calls for interdisciplinary collaboration and ethical regulation to ensure responsible AI adoption, envisioning a future where AI continues to drive innovation and productivity across domains [40].

The convergence of these deployment and maintenance focused studies suggests that AI is increasingly being woven into the operational backbone of software systems rather than being confined to discrete development tasks. This operational integration, often described under the umbrella term AIOps, extends the reach of artificial intelligence beyond the boundaries of the traditional Software Development Life Cycle and into the ongoing management of live production systems. As organizations come to rely on AI not only to build software but also to keep it running reliably, questions of explainability and accountability, already raised earlier

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

in this paper, take on additional urgency, since failures in an automated monitoring or self healing system can have immediate and visible consequences for end users.

Kokol et al. [41] gives a comprehensive synthetic knowledge synthesis of research on artificial intelligence (AI) in software engineering (SE) compiled by Kokol et al. [41]. Through a bibliometric analysis map including over 9,000 publications identified between the years 2019 through 2024, the study reveals research themes, including the use of natural language processing in software engineering, artificial intelligence applied to the management process of the software development life cycle, machine learning-related research focusing on fault prediction/estimation, the use of deep learning in intelligent code management, mining software repositories to enhance software quality. Results indicate China has the heaviest publication output, trailed closely by the United States and India with major funding provided from their respective national science foundations. However, all research is yet in the 'maturation' stage with a predominance of publication from conferences. The article points out how fuzzy logic, NLP, machine learning, and deep learning are being used in AI for automating testing, improving requirement engineering, defect prediction, estimation of development effort, and understanding code. The article discusses challenges in AI and SE integration like ethics, technical debt, and effects of COVID-19 on co-development in software development. The integration of AI and SE is demonstrated in this article as it increases quality, decreases development cost, and increases developer happiness [42].

Viewed as a whole, the literature surveyed in this section reveals a field that is simultaneously mature and unevenly developed. Mature, in the sense that individual applications of AI, ranging from code completion to defect prediction to anomaly detection, have each accumulated a substantial and methodologically diverse body of supporting research. Unevenly developed, in the sense that this research effort has not been distributed equally across the phases of the Software Development Life Cycle, with implementation and testing attracting considerably more attention than requirements engineering or long term maintenance. This unevenness in the academic literature mirrors, and arguably helps to explain, the unevenness in practitioner adoption that the present study sets out to document empirically. A further observation that emerges from this review is methodological in nature: a large proportion of the cited studies rely on case studies, proof of concept implementations, or controlled experiments conducted within a single organization or research laboratory, with comparatively few studies drawing on cross organizational survey data of the kind collected in this study. This gap further motivates the descriptive survey methodology adopted here, which is described in detail in the following section.

Table 2.1 shows the overall summary of key contributions of authors.

Table 2.1 key Contributions Summary

SDLC Phase	Ref	Methodology	Key Contribution
Requirements & Design	[43]	NLP-driven framework using transformer models	Identifies ambiguities in requirements; improves traceability and reduces errors.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

	[16]	AI system using CNNs and GNNs	Recommends optimal design patterns based on historical data and system constraints.
	[17]	Reinforcement learning integration	Iteratively refines software architecture models in response to changing requirements.
	[18]	Comprehensive review, case studies, survey (n=250)	Highlights AI's role in requirement gathering and intelligent system modeling; proposes a unified framework for responsible AI.
Coding & Implementation	[13]	Empirical evaluation of GitHub Copilot	Shows 35–50% faster task completion with maintained quality; reduces boilerplate code.
	[11]	Fine-tuned CodeBERT models for embedded systems	Generates secure and efficient code snippets; emphasizes domain-specific training.
	[23]	Multi-agent AI framework	Enables collaborative code generation, improving consistency in large-scale projects.
	[24]	AI-SEAL taxonomy development and application	Classifies AI use in SE by application point, AI type, and automation level for risk assessment.
Testing & Quality Assurance	[44]	ML-based test case prioritization using historical data	Reduces regression testing time by up to 40% through risk-based module prediction.
	[45]	Generative Adversarial Networks (GANs)	Synthesizes realistic test inputs to improve edge-case coverage in safety-critical systems.
	[46]	Interpretable deep learning model (XAI)	Predicts defects and provides actionable root-cause insights for proactive bug prevention.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

	[47]	Systematic Literature Review (55 studies, 1990–2019)	Maps SE practices for AI/ML systems to SWEBOK; highlights testing, ethics, and reproducibility challenges.
Deployment & Maintenance	[48]	LSTM networks for predictive maintenance	Predicts failures in cloud-based microservices to enable preemptive scaling and minimize downtime.
	[49]	Unsupervised learning for anomaly detection	Reduces MTTR by 30% through real-time monitoring of runtime logs.
	[50]	Reinforcement learning-based self-healing framework	Automates patching and maintenance to address vulnerabilities and performance degradation.
	[51]	Mixed-methods: literature review, surveys, expert interviews	Explores AI's broader impact; emphasizes explainable AI and ethical regulation across domains.
	[52]	Bibliometric mapping & content analysis (9000+ publications, 2019–2024)	Identifies major AI-SE themes (e.g., NLP, fault prediction, code management) and ethical concerns.

The summary presented in Table 2.1 underscores the breadth of existing research while also highlighting the comparatively thin coverage of requirements engineering and maintenance relative to coding and testing. Building on these observations, the following section outlines the research methodology developed for this study, which is designed specifically to capture practitioner level evidence across all six phases of the Software Development Life Cycle within a single, internally consistent dataset.

3. Research Methodology

This study employed a descriptive survey-based approach to investigate the integration of Artificial Intelligence (AI) across the Software Development Life Cycle (SDLC). The methodology was designed to collect and analyze data from professionals in the field, providing a quantitative and qualitative snapshot of current practices and perceptions. Figure 3.1 outlines the overall research workflow.
A survey based design was selected over alternative methodologies, such as in depth case studies or controlled experiments, because it offers a practical means of capturing breadth across the full Software Development Life Cycle within the

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

resource constraints of this study. Case studies and experiments, while capable of producing rich, detailed insight into a single organization or tool, are less well suited to answering questions about how adoption varies across many different phases and organizational contexts simultaneously. A descriptive survey, by contrast, allows data to be gathered efficiently from a geographically and organizationally diverse pool of respondents, enabling the kind of cross phase comparison that RQ1 and RQ2 require. This methodological choice does carry trade offs, which are addressed explicitly in the limitations discussed later in this section, but it was judged the most appropriate fit for the exploratory, broad based aims of this research.

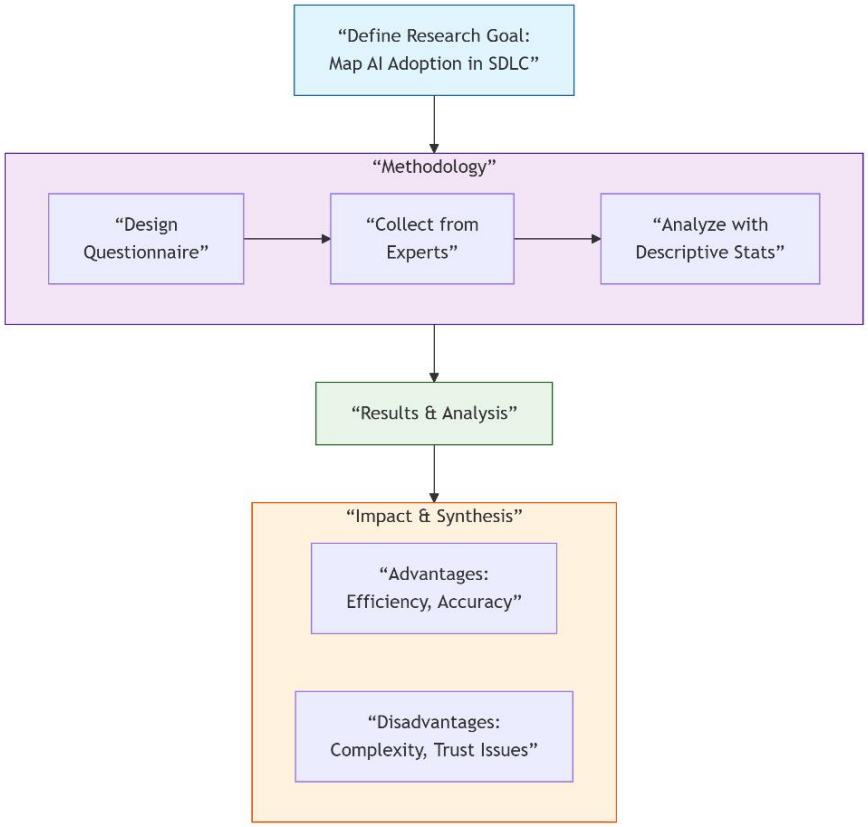


Figure 3.1: Research Methodology Diagram

3.1 Research Design and Instrument

A structured online questionnaire served as the primary data collection instrument. Its development followed a three-stage validation process:

This three stage validation process was adopted to maximize both the content validity and the practical usability of the survey instrument. Relying solely on a literature derived item pool risks producing questions that are academically comprehensive but poorly matched to the vocabulary and priorities of practicing engineers, while relying solely on practitioner input risks omitting theoretically important constructs that respondents might not think to mention spontaneously. Combining the two sources of input, and then subjecting the resulting draft to both

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

expert review and pilot testing, was intended to produce an instrument that was simultaneously grounded in the academic literature and intelligible to the working professionals who would ultimately complete it.

- **Item Generation:** Initial questions were derived from a literature review focused on AI tools and applications in key SDLC phases (Requirement, Design, Implementation, Testing, Deployment, Maintenance).
- **Expert Review:** A draft was assessed by three senior software engineering researchers for clarity, relevance, and coverage.
- **Pilot Testing:** The survey was refined based on feedback from a pilot group (n=10) of target professionals to ensure technical accuracy and appropriate completion time.

The final instrument comprised four sections:

- **Section A:** Demographic and professional background.
- **Section B:** Likert-scale items measuring the frequency and intensity of AI tool usage for specific tasks in each SDLC stage.
- **Section C:** Questions evaluating perceived impacts (e.g., on efficiency, accuracy, complexity).
- **Section D:** Open-ended questions for qualitative insights on challenges and future trends.

This four part structure was designed to balance breadth and depth within a single instrument. Section A establishes the demographic and organizational context necessary to interpret later responses, Sections B and C provide the quantitative core of the survey from which the descriptive statistics reported in the results section are derived, and Section D supplies qualitative depth that the closed ended items in Sections B and C cannot capture on their own. Including an open ended section was considered particularly important given the exploratory nature of RQ3, which concerns organizational and ethical issues that are not yet well enough understood to be reduced to a fixed set of closed ended response options.

3.2 Data Collection

The target population consisted of software engineers, developers, project managers, and researchers with hands-on experience in AI-enabled software projects. A **purposive sampling** technique was used to reach qualified experts through professional networks (e.g., LinkedIn, relevant online communities, and research forums). Data was collected over an eight-week period via an online platform, yielding **31 valid and complete responses** for analysis.

Purposive sampling was selected over probability based sampling methods because the population of professionals with substantial, hands on experience using AI tools throughout the Software Development Life Cycle is not easily enumerated through any existing sampling frame, such as a company directory or professional registry. Reaching this specific population required deliberately targeting individuals and communities already engaged with AI enabled development practices, rather than sampling at random from the broader population of software professionals, many of whom would have had little or no relevant experience to draw on. While this approach necessarily limits the statistical generalizability of the findings, a limitation acknowledged explicitly later in this section, it was judged the most practical means

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

of obtaining data from respondents genuinely qualified to speak to the questions under investigation.

3.3 Data Analysis

Collected data was analyzed using **descriptive statistics** to quantify central tendencies and distributions. Analysis was performed with statistical software (e.g., SPSS/R) and involved:

- Calculating **mean scores and standard deviations** for adoption levels in each SDLC phase to enable direct comparison.
- Generating **frequency distributions and cross-tabulations** to explore relationships between adoption levels and demographic variables.
- Conducting a **thematic analysis** on responses from open-ended questions to identify recurring themes related to barriers, benefits, and future directions.

Combining quantitative and qualitative analysis in this way was intended to support a form of methodological triangulation, in which patterns identified through descriptive statistics could be cross checked against, and enriched by, the thematic content of open ended responses. Where the two forms of evidence converge, as they do in several places reported in the results section, this convergence strengthens confidence in the underlying finding. Where they diverge, or where the qualitative data surfaces nuance not captured by the quantitative items, this divergence is treated as an opportunity for deeper interpretation rather than as a methodological inconsistency to be resolved away.

3.4 Ethical Considerations

All participants provided informed consent prior to survey commencement. The study ensured **anonymity and confidentiality**; no personally identifiable information was collected, and data was aggregated for reporting purposes.

These ethical safeguards were considered essential given the relatively candid nature of several open ended responses, particularly those touching on trust in AI generated outputs and dependency concerns, which respondents might have been reluctant to share had their identities or organizational affiliations been at risk of disclosure. By guaranteeing anonymity at the outset and by reporting only aggregated findings throughout this paper, the study aimed to encourage honest reflection rather than socially desirable responses that might have understated the genuine challenges respondents experience in their daily work.

3.5 Limitations

The study acknowledges certain limitations:

- The use of non-probability sampling may affect the generalizability of findings.
- Data is based on self-reported perceptions, which may include subjective bias.
- The cross-sectional design captures a snapshot in time rather than longitudinal trends.

Each of these limitations is revisited in more detail in the dedicated limitations section following the presentation of results, where their implications for interpreting specific findings are considered explicitly. Readers are encouraged to bear these constraints in mind throughout the results section that follows,

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

particularly when considering the statistical precision implied by percentages calculated from a sample of 31 respondents.

4. Results and Discussion

This section presents and discusses the findings from the survey of 31 software development professionals. The results are structured to analyze: (1) demographic distribution, (2) AI adoption levels across the SDLC, (3) perceived impact on performance metrics, (4) quantified adoption challenges, and (5) qualitative insights on barriers.

Each subsection below follows a consistent presentation format in which descriptive statistics are first reported and visualized, followed by a short discussion that interprets the practical significance of the pattern observed. This format is intended to keep the empirical evidence and its interpretation clearly separated, allowing readers to evaluate the underlying data independently of the interpretive claims advanced in this paper.

4.1 Respondent Demographics

The survey captured a diverse cross-section of the software industry, ensuring that the findings reflect varied perspectives. The primary roles of respondents are distributed as shown in Figure 1, with Software Developers/Engineers (32%) and AI/ML Engineers (26%) forming the majority, providing a strong foundation of technical expertise. The remaining respondents included Quality Assurance professionals (19%), Architects/Technical Leads (13%), Development Managers (10%), and DevOps Engineers (10%).

Experience and Organization Size: The cohort exhibited a balanced mix of experience levels, from early-career professionals (1-3 years: 42%) to seasoned experts (13+ years: 26%). This distribution ensures that insights capture both modern tool adoption trends and seasoned, strategic perspectives. Respondents worked across a spectrum of organization sizes, from nimble Startups (42%) to Global Corporations (19%), indicating that the findings are not skewed toward a specific organizational context.

This combination of varied experience levels and varied organizational contexts is particularly valuable for a study concerned with adoption patterns, since both factors are plausible drivers of differences in AI usage. More experienced respondents may bring a longer view of how development practices have evolved over time, allowing them to situate current AI tools within a broader history of tooling change, while early career respondents may have entered the profession at a point when AI assisted development was already the norm, shaping their expectations and habits differently from the outset. Similarly, respondents in small startups often described workflows in which a single engineer might be responsible for tasks spanning several SDLC phases, a circumstance that could plausibly encourage broader, if shallower, AI tool adoption across phases, whereas respondents in larger organizations more frequently described specialized roles in which AI adoption decisions were concentrated within particular teams, such as a dedicated QA function or a platform engineering group.

Figure 4.1 shows the distribution of survey respondents by primary role.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

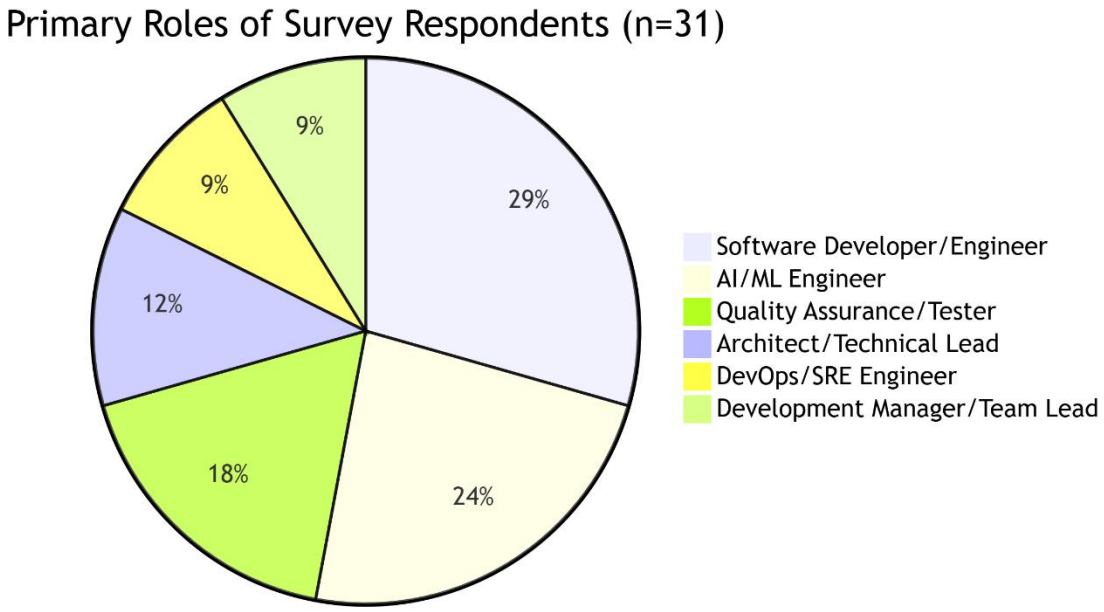


Figure 4.1: Distribution of survey respondents by primary role.

Table 4.1 shows the overall questionnaire counting.

Table 4.1: Questionnaire Counting

Category	Subcategory	Count	Percentage
Primary Role	Software Developer/Engineer	10	32%
	AI/ML Engineer	8	26%
	Quality Assurance/Tester	6	19%
	Architect/Technical Lead	4	13%
	Development Manager/Team Lead	3	10%
	DevOps/SRE Engineer	3	10%
Experience	1-3 years	13	42%
	4-7 years	6	19%
	8-12 years	4	13%
	13+ years	8	26%
Organization Size	Startup (1-50)	13	42%
	SME (51-500)	8	26%
	Large (501-5,000)	4	13%
	Global (5,000+)	6	19%

4.1 AI Adoption across SDLC phases

The integration of AI is highly uneven across the Software Development Life Cycle (SDLC), revealing a clear pattern of tool maturity and practical utility. As illustrated

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

in Figure 2, adoption is predominantly concentrated in the latter, execution-oriented phases.

High Adoption Phases: The Coding & Implementation phase leads with 67% of teams reporting "Extensive" or "Moderate" use, driven by the proliferation of AI-powered code completion, generation, and debugging tools (e.g., GitHub Copilot). Similarly, the Testing & Quality Assurance phase shows 61% high/moderate adoption, where AI is leveraged for test case generation, predictive defect analysis, and automated script maintenance.

Moderate Adoption Phases: Maintenance & Evolution (55% high/moderate) and Design & Architecture (52%) show emerging use cases in code refactoring, architectural pattern suggestions, and technical debt identification. Deployment & Operations (48%) indicates cautious integration for CI/CD pipeline optimization and monitoring.

Low Adoption Phase: Strikingly, Requirements Engineering has the lowest adoption at 40%. This indicates a significant gap where strategic, human-centric tasks like stakeholder need elicitation, prioritization, and specification remain less amenable to current AI automation.

Discussion: This adoption gradient underscores a market and research focus on augmenting *how* software is built rather than *what* should be built. The high coding/testing adoption reflects readily available, task-specific tools, while the low requirements adoption highlights the complexity of automating cognitive and collaborative processes involving ambiguity and social negotiation.

This pattern also has implications for how the benefits of AI adoption are distributed across a development organization. Because coding and testing tools tend to be adopted earlier and more readily, the productivity gains associated with AI are likely to accrue disproportionately to developers and quality assurance staff in the near term, while business analysts, product owners, and other roles concentrated in the requirements phase may see comparatively little direct benefit from current generation tools. Over time, as natural language processing techniques continue to mature and as organizations gain confidence in applying AI to less structured, more judgment intensive tasks, this gap may narrow. For the present, however, the data gathered in this study suggest that organizations seeking to maximize the return on their investment in AI tooling should not assume that gains observed in coding or testing will automatically generalize to earlier, more strategic phases of the Software Development Life Cycle.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

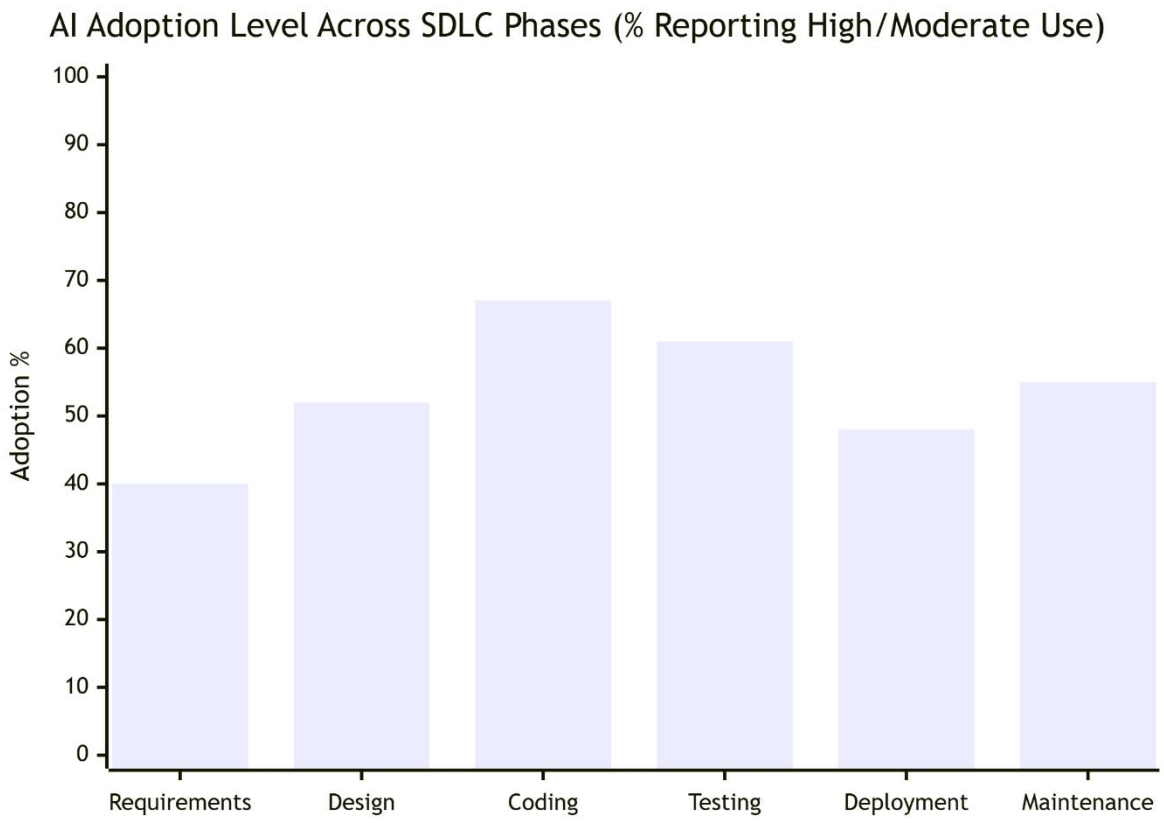


Figure 4.2: AI adoption intensity across SDLC phases, measured by the percentage of respondents reporting "Extensive" or "Moderate" use.).

4.2 Perceived Impact on Performance Metrics

Respondents perceived a strongly positive impact from AI tools across key performance metrics, as shown in Figure 4.3. The highest-rated impact was on Development Cycle Time (Avg: 4.03/5), directly correlating with the automation of repetitive coding and testing tasks. Developer Productivity (3.90/5) followed closely, emphasizing efficiency gains.

Notably, impacts on Code Quality and Team Collaboration were rated identically (3.83/5). This suggests that AI tools contribute to quality through consistent style adherence and error detection, while also fostering collaboration via automated documentation and shared code understanding. The relatively lower score for Risk Management (3.50/5) indicates that AI's role in proactive risk identification and mitigation is still underdeveloped or under-trusted.

Discussion: The results confirm AI as a potent efficiency driver but also reveal its current limitations in higher-order strategic assurance. The parallel scores for quality and collaboration point to an indirect but valuable social benefit of AI tooling within teams.

These performance impact findings should be read alongside the adoption data presented immediately above them, since the two together suggest a coherent story.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

Phases where adoption is highest, namely coding and testing, are also the phases most directly implicated in the cycle time and productivity gains that respondents rated most favorably, indicating a plausible link between concentrated tool adoption and the specific performance metrics that respondents associate with that adoption. The comparatively muted rating for risk management, by contrast, is consistent with the relatively low adoption of AI in requirements engineering and architectural design reported earlier, phases that are most directly concerned with anticipating and mitigating project level risk. This consistency across independent survey items lends additional credibility to the overall pattern of findings reported in this section.

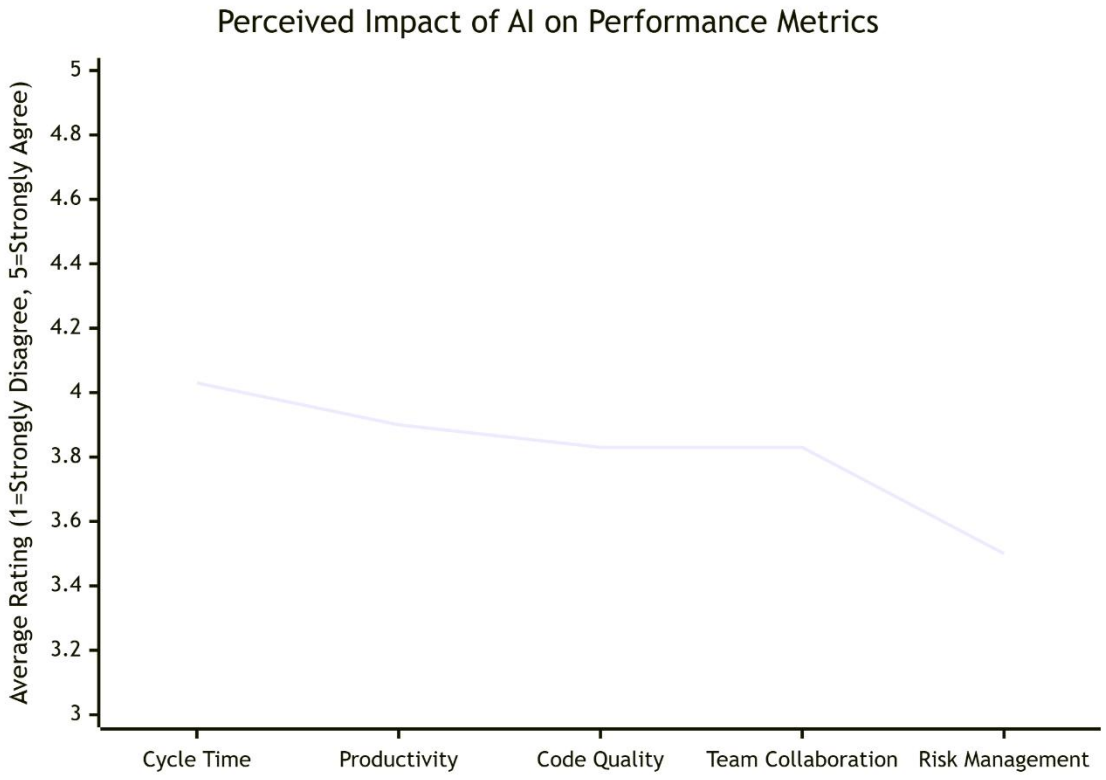


Figure 4.3: Average Likert-scale ratings for the perceived impact of AI on software development performance metrics.

4.3 Challenges in AI Adoption

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

The adoption of AI is impeded by several significant challenges, ranked by severity in Figure 4. Data Quality and Availability (3.87/5) emerged as the foremost obstacle, highlighting the foundational "garbage in, garbage out" principle that limits AI effectiveness in real-world, often messy, development environments. Skill Gaps (3.73/5) and Integration Complexity (3.63/5) were the next major hurdles, pointing to a dual need for workforce upskilling and more seamless tool interoperability with existing development ecosystems. Concerns regarding Model Explainability (XAI) and Security Risks were rated as moderate but substantial (3.53/5 and 3.50/5, respectively).

Discussion: The prominence of data and skill challenges suggests that organizational and infrastructural readiness is as critical as the tools themselves. Interestingly, Organizational Resistance was rated the lowest (3.17/5), implying that cultural pushback is less of a barrier than the practical, technical difficulties of implementation.

It is also worth noting how closely these challenge ratings track the concerns raised in the literature reviewed earlier in this paper. The prominence of data quality as the leading challenge echoes the data dependency concerns documented across multiple studies cited in the literature review, while the comparatively high ratings for skill gaps and integration complexity reinforce the economic and organizational hurdles to AI adoption discussed in the introduction. The relatively low score for organizational resistance is, in this light, a genuinely novel finding rather than a simple restatement of the existing literature, since much of the popular discourse around AI adoption emphasizes cultural reluctance as a primary barrier. The survey data collected here suggest that, at least among the practitioners surveyed, the practical mechanics of implementation, namely data readiness, skill availability, and tool interoperability, present a more immediate obstacle than resistance rooted in organizational culture.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

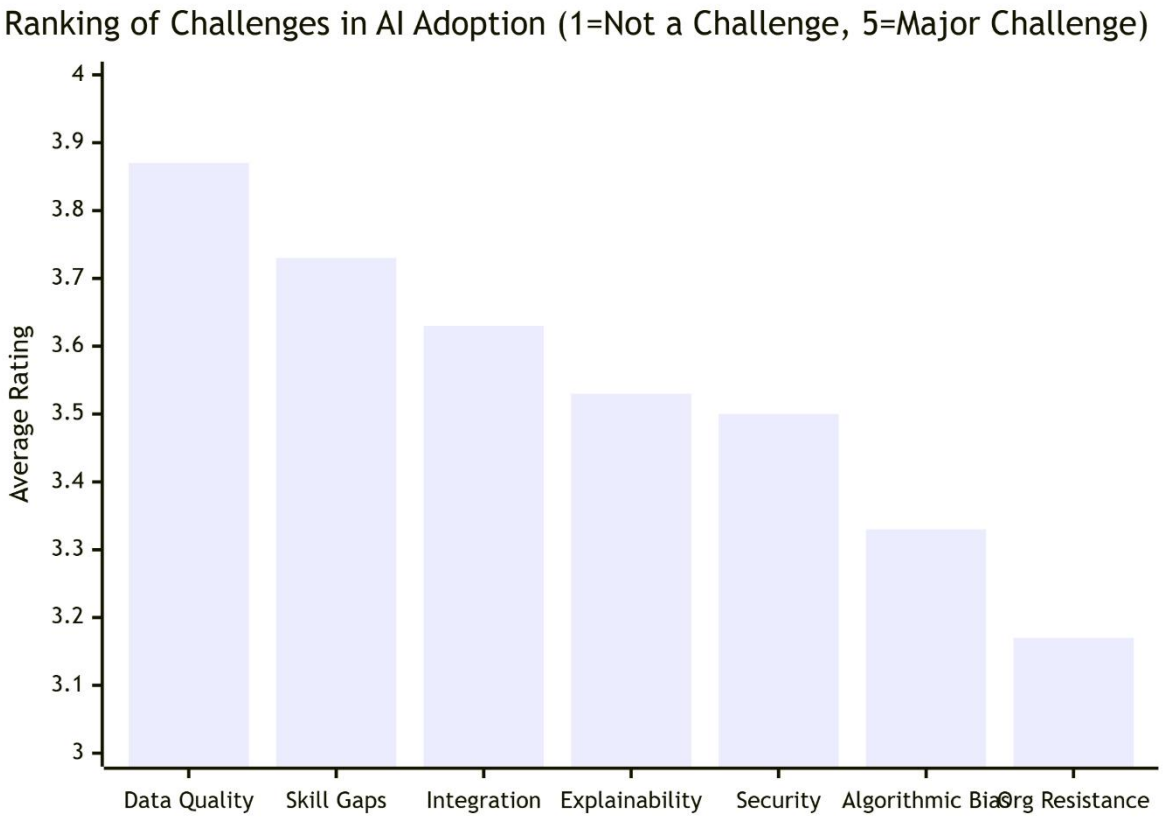


Figure 4.4: Severity of challenges faced during AI adoption in software engineering.

4.4 Qualitative Insights on Barriers

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

When asked about the biggest problem in increasing the application of artificial intelligence in their work processes, the following themes emerged:

- **Trust and Reliability Issues:** Some participants, especially those in QA/testing roles, highlighted trust issues. A QA engineer said, "As a QA, the one major barrier I would fix to adopt more AI in my workflow is lack of trust in AI outputs." Another participant added, "Unclear reliability – Users still have to double-check outputs, which reduces time savings."
- **Data Quality Problems:** AI/ML experts mentioned data-related problems. A comment reads: "The biggest problem is data silos and data quality, which requires manual cleaning and lack of context." A Development Manager explained: "In QA processes, AI is only as good as the data it is trained on. Without high-quality data, AI-based test coverage is no longer reliable."
- **Skill and Dependency Risks:** "Using more and more AI in workflow can cause the person to be more dependent on it rather than his/her own knowledge" is what some respondents have expressed concern about.
- **Clarity and Explainability:** A Software Developer said, "If I could change one thing that would make a big difference, I would pick clarity in problem understanding."

Read together, these four qualitative themes, trust and reliability, data quality, skill dependency, and clarity and explainability, paint a picture of AI adoption that is shaped as much by human and organizational factors as by the underlying technical capability of the tools themselves. None of the concerns raised by respondents centers on whether AI tools are capable of performing useful work; rather, each concern centers on the conditions under which that capability can be trusted, integrated, and understood well enough to be relied upon in practice. This distinction is significant for organizations designing AI adoption strategies, since it suggests that investment in tool capability alone is unlikely to be sufficient. Complementary investment in data governance, workforce training, and explainability practices may be equally important in determining whether AI adoption translates into the kind of sustained, balanced integration across the Software Development Life Cycle that this study set out to investigate.

Figure 4.5 shows the Qualitative barriers.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

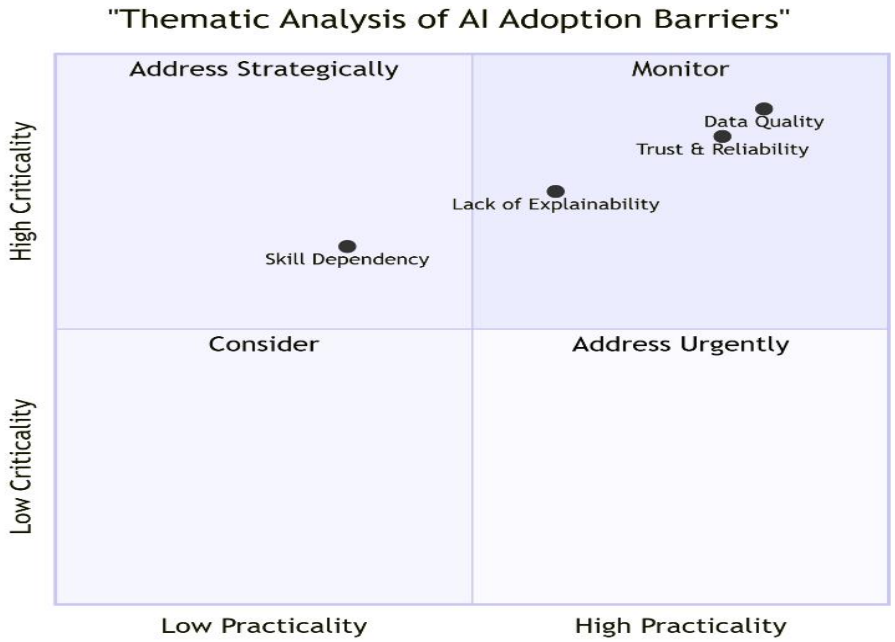


Figure 4.5: Qualitative barriers mapped by their perceived criticality and practical immediacy, based on thematic analysis of open-ended feedback.

The important user feedback related to each barrier is presented in Table 4.2.
Table 4.2: key User Feedback

Barrier	**Key User Feedback**
Trust & Reliability	"Lack of trust in AI outputs.", "Users still have to double-check results."
Data Quality	"AI is only as good as the data it learns from.", "Poor data quality and data silos limit AI."
Skill Dependency	"Overuse of AI can make users dependent."
Explainability	"Lack of clarity in problem understanding."

The qualitative and quantitative findings presented across this section together provide a detailed empirical answer to the three research questions posed at the outset of this paper. The following section considers the limitations of this evidence in more detail and outlines directions for future research that could extend and strengthen the conclusions drawn here.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

5. Limitations and Future Work:

This research has a number of limitations. Firstly, the sample size (N = 31), although diverse, cannot be generalized for all software development environments. Secondly, the use of self-reported data may introduce social desirability bias. Thirdly, the cross-sectional approach taps into attitudes at a point in time in a rapidly changing area.

Taken together, these limitations suggest that the findings reported in this paper are best understood as an initial, exploratory mapping of AI adoption patterns rather than as a definitive or statistically representative account of the software industry as a whole. This characterization is not intended to diminish the value of the findings, since exploratory studies of this kind play an important role in generating hypotheses and identifying patterns worthy of more rigorous follow up investigation, but it does counsel caution in extrapolating the specific percentages reported here to organizations or contexts not represented in the sample.

For future studies, the following directions are recommended: (1) longitudinal studies to track adoption trends over time, (2) studies on the relationship between particular AI tools and performance, (3) studies on the relationship between organizational size/maturity and adoption trends, and (4) empirical studies on the relationship between AI adoption and software quality metrics through code analysis. Each of these four recommended directions addresses a specific limitation of the present study. Longitudinal research would address the cross sectional nature of the current design by tracking how individual organizations' adoption patterns evolve as tools mature and as practitioners gain experience. Studies linking particular tools to particular performance outcomes would move beyond the general, perception based impact ratings collected here toward more objective, tool specific measures of effect. Research connecting organizational size and maturity to adoption trends would help determine whether the patterns observed in this predominantly startup heavy sample generalize to larger, more established organizations. Finally, empirical studies grounded in direct code analysis, rather than self reported perception, would provide a valuable external check on the relationship between AI adoption and software quality that respondents in this study reported based on their own subjective impressions.

Future Directions

Looking ahead, the future of software engineering is inextricably linked to the evolution of artificial intelligence tools and methodologies[16].

Some of the most important trends that are emerging include a change in the nature of developer work, where experts are expected to move from code writing to managing and validating AI-generated content [17]. New job roles, such as Prompt Engineers and AI Model Validators, are also expected to emerge as a necessity in the future [13]

This anticipated shift in developer responsibility carries significant implications for software engineering education and professional development. Curricula that have traditionally emphasized syntax mastery and algorithmic problem solving as core competencies may need to place comparatively greater emphasis on skills such as critically evaluating AI generated output, recognizing the boundary conditions under which a model is likely to perform poorly, and effectively communicating

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 8 (2025)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

intent to an AI system through carefully constructed prompts. Universities and professional training programs that adapt early to this shift may produce graduates better prepared for the AI augmented development environments described throughout this paper, while those that do not risk producing graduates whose skills are mismatched with the tools they will encounter immediately upon entering the workforce.

- There will also be a strong focus on Trustworthy AI (TAI), which will emphasize the creation of systems that are secure, explainable, fair, and aligned with societal values[18].
- Research is gradually moving towards self-contained software development, with a vision of Level 5 Autonomy where AI can independently handle the entire software development lifecycle (SDLC), from comprehending high-level requirements to deployment and maintenance, including self-healing and self-improvement capabilities [15].
- Moreover, there will be a need for new architectures and approaches, such as sound MLOps infrastructure and AI development platforms, to efficiently handle the life cycle of models that are deployed in software systems. This will include governance, monitoring for model drift, and retraining to ensure continued relevance and performance[19].

The cumulative effect of these emerging trends, changing developer roles, an increased emphasis on trustworthy and explainable AI, progress toward greater autonomy, and the maturation of MLOps infrastructure, is likely to be a software engineering discipline that looks substantially different a decade from now than it does today. Whether this transformation proceeds smoothly or unevenly will depend heavily on whether the organizational, ethical, and skill related challenges identified throughout this paper are addressed proactively or are instead allowed to accumulate as unmanaged technical and organizational debt.

This paper aims to offer a holistic discussion on this revolutionary integration. By conducting a systematic review of the existing literature, an innovation and advantage analysis, a critical discussion on the challenges, and a future outlook discussion, this paper aims to offer a holistic understanding of how AI is revolutionizing software engineering. In addition, a research methodology will be proposed to empirically study the adoption, effects, and challenges of AI tools in an industrial context, thus laying a foundation for future research and innovation in this ever-evolving field.

With this broader context established, the paper now turns to its concluding section, which synthesizes the empirical findings reported above into a coherent set of takeaways for both researchers and practitioners.

6. Conclusion

This research examines the integration of Artificial Intelligence over the Software Development Life Cycle (SDLC) and evaluates the adoption levels of AI over various phases of the SDLC. The findings reveal that Artificial Intelligence serves as an effective enabler in modern software development, with extensive adoption being noticed over the implementation and testing phases of software development, where automation tools, code analyzers, and defect detection tools are extensively used. The other phases of software development, such as requirements analysis, design,

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

and maintenance, have relatively lower adoption levels of AI.

This pattern of uneven adoption, concentrated in execution oriented phases and comparatively sparse in strategic, judgment intensive phases, represents perhaps the single most important empirical finding of this study. It indicates that, despite more than a decade of rapid progress in artificial intelligence research, the practical integration of AI into software engineering remains at an early and unevenly distributed stage of maturity, with substantial room for growth in precisely those phases, requirements engineering and long term maintenance, that are often most consequential for a project's ultimate success or failure.

Despite the obvious benefits of AI, such as increased productivity, better software quality, and shorter development cycles, the paper has identified some challenges, such as integration complexity, lack of qualified personnel, and issues related to the reliability and trustworthiness of decisions made by AI. These challenges need to be addressed in order to achieve more balanced and effective adoption of AI throughout all stages of the SDLC.

In conclusion, the results emphasize the growing importance of AI in software development and the need for its effective implementation and development. Future research can focus on developing more explainable, interoperable, and domain-specific AI solutions to ensure smooth integration at all stages of the SDLC.

Ultimately, the path toward more balanced AI integration across the Software Development Life Cycle will depend on coordinated progress across several fronts simultaneously: continued technical innovation to extend AI capability into less structured, more judgment intensive tasks; sustained investment in data quality and organizational readiness; deliberate workforce development to close existing skill gaps; and the gradual establishment of governance practices capable of keeping pace with a rapidly evolving technology. The findings reported in this paper are offered as a contribution toward that broader, ongoing effort.

7 References

1. Alenezi, M. and M. Akour, *Ai-driven innovations in software engineering: a review of current practices and future directions*. Applied Sciences, 2025. **15**(3): p. 1344.
2. Rajuroy, A., W. Liang, and B.J. Mary, *Revolutionizing Software Development with AI: The Path to 2030*.
3. Ajiga, D., et al., *Enhancing software development practices with AI insights in high-tech companies*. IEEE Software Engineering Institute, Technical Report TR-2024-003, 2024.
4. Müller, M., *Advancements in Software Engineering through Artificial Intelligence*. 2024.
5. Bailey, L., *The Impact of AI on Software Development*. 2024, Doctoral dissertation, Worcester Polytechnic Institute.
6. Khan, A. and A.A.A. Jilani, *From Algorithms to Intelligence: The Historical Perspective of AI in Software Development*, in *Generative AI in Software Engineering*. 2025, IGI Global Scientific Publishing. p. 1–56.
7. Abbas, T., et al., *Enhancing Software Engineering With AI: Innovations, Challenges, and Future Directions*. IET Software, 2025. **2025**(1): p. 5691460.

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

8. Zadeh, E.K., A.B. Khoulenjani, and M. Safaei, *Integrating AI for agile Project Management: Innovations, challenges, and benefits*. International Journal of Industrial Engineering and Construction Management (IJIECM), 2024. **1**(1): p. 1–10.
9. Rajuroy, A., W. Liang, and B.J. Mary, *AI-Driven Collaboration in Software Engineering: Enhancing Productivity and Innovation*.
10. Siddiqui, N., *The role of AI integration in fostering innovation within software houses*. 2024.
11. Boddy, S., et al., *Computer integrated construction: A review and proposals for future direction*. Advances in Engineering Software, 2007. **38**(10): p. 677–687.
12. Nguyen-Duc, A., et al., *Generative artificial intelligence for software engineering A research agenda*. Software: Practice and Experience, 2025. **55**(11): p. 1806–1843.
13. Jantunen, M., *Integration of Artificial Intelligence Ethics into Software Engineering Processes: Challenges, Concerns, and Opportunities*. JYU Dissertations, 2025.
14. Mohammad, A. and B. Chirchir, *Challenges of integrating artificial intelligence in software project planning: a systematic literature review*. Digital, 2024. **4**(3): p. 555–571.
15. Benitez, C.D. and M. Serrano, *The integration and impact of artificial intelligence in software engineering*. International Journal of Advanced Research in Science, Communication and Technology (IJARSCT) International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal, 2023. **3**(2).
16. Pai, A., et al. *Integrating AI into Software Engineering: A Critical Review and Future Directions*. in *2025 International Conference on Intelligent Control, Computing and Communications (IC3)*. 2025. IEEE.
17. Durrani, U.K., et al., *A decade of progress: A systematic literature review on the integration of AI in software engineering phases and activities (2013-2023)*. IEEE Access, 2024. **12**: p. 171185–171204.
18. Nyaga, F., *AI-Driven Software Engineering: A Systematic Review of Machine Learning's Impact and Future Directions*. 2025.
19. Lefèvre, É., *AI-Based Modeling in Software Engineering: Techniques, Applications, and Future Directions*. International Journal of Trend in Research and Development, 2022. **9**(5).
20. Chen, X., et al., *Deep learning-based software engineering: progress, challenges, and opportunities*. Science China Information Sciences, 2025. **68**(1): p. 111102.
21. Arif, H., A.K.S. Ali, and H.A. Nabi, *IoT Security through ML/DL: Software Engineering Challenges and Directions*. ICCK Journal of Software Engineering, 2025. **1**(2): p. 90–108.
22. Gao, C., et al., *The current challenges of software engineering in the era of large language models*. ACM Transactions on Software Engineering and Methodology, 2025. **34**(5): p. 1–30.
23. Deng, Y., et al. *Leveraging nlp in finance: A synergistic approach using large language models and chain-of-thought reasoning*. in *Proceedings of the 5th International Conference on Artificial Intelligence and Computer Engineering*. 2024.
24. Kalusivalingam, A.K., et al., *Enhancing AI-Driven Pathology Image Analysis Using Convolutional Neural Networks and Transfer Learning Techniques*. International

Online ISSN 3007-3197
Print ISSN 3007-3189

Journal of AI and ML, 2013. **2**(10).
25. Mamun, M.N.H., *Integration Of Artificial Intelligence And DevOps In Scalable And Agile Product Development: A Systematic Literature Review On Frameworks*. ASRC Procedia: Global Perspectives in Science and Scholarship, 2024. **4**(1): p. 01–32.
26. Li, S., et al. *Assessing the performance of ai-generated code: A case study on github copilot*. in *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. 2024. IEEE.
27. Li, Z., et al., *Fine-Tuning Code Language Models to Detect Cross-Language Bugs*. arXiv preprint arXiv:2507.21954, 2025.
28. Liu, A., et al., *RA-Gen: A Controllable Code Generation Framework Using ReAct for Multi-Agent Task Execution*. arXiv preprint arXiv:2510.08665, 2025.
29. Wischnewski, M., et al., *In seal we trust?: Investigating the effect of certifications on perceived trustworthiness of AI systems*. Human-Machine Communication, 2024. **8**: p. 141–162.
30. Kumar, H. and V. Saxena, *Machine Learning for Test Case Prioritization in Continuous Integration: A Comprehensive Analysis*. Int. J. Advanced Networking and Applications, 2024. **15**(06): p. 6218–6228.
31. Reyes, A., et al., *Calibration algorithms assessment for soil nitrogen prediction with near-infrared spectroscopy and data augmentation*. Agrociencia, 2024: p. 1–15.
32. Zhou, R., et al., *A model-based deep learning approach to interpretable impact force localization and reconstruction*. Mechanical Systems and Signal Processing, 2025. **224**: p. 111977.
33. Nascimento, E., et al., *Software engineering for artificial intelligence and machine learning software: A systematic literature review*. arXiv preprint arXiv:2011.03751, 2020.
34. Soares, E., et al., *The effects of continuous integration on software development: a systematic literature review*. Empirical Software Engineering, 2022. **27**(3): p. 78.
35. Kumar, I., B.K. Tripathi, and A. Singh, *Attention-based LSTM network-assisted time series forecasting models for petroleum production*. Engineering Applications of Artificial Intelligence, 2023. **123**: p. 106440.
36. Verma, O., S. Verma, and T. Perumal, *Advancement of intelligent computational methods and technologies*. 2024: CRC Press.
37. Inshi, S., et al. *Enhancing autonomy of context-aware self-healing in fog native environments*. in *International Symposium on Foundations and Practice of Security*. 2024. Springer.
38. Costa, A.C.M., et al., *Impact of Generative Artificial Intelligence on Knowledge Management in Software Engineering: A Systematic Mapping Study*. Simpósio Brasileiro de Qualidade de Software (SBQS), 2025: p. 55–66.
39. Abrahão, S., et al., *Software Engineering by and for Humans in an AI Era*. ACM Transactions on Software Engineering and Methodology, 2025. **34**(5): p. 1–46.
40. Figueiredo, A.C., R. Pereira, and M.Â. da Silva, *Exploring the Integration of Artificial Intelligence and DevOps for Agile Product Development*, in *Digital Technologies and Transformation in Business, Industry and Organizations*. 2025, Springer. p. 27–39.

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

41. Kokol, P., *Synthetic knowledge synthesis in hospital libraries*. Journal of Hospital Librarianship, 2024. **24**(1): p. 10–17.
42. Kokol, P., et al., *Evolutionary Game Theory Use in Healthcare: A Synthetic Knowledge Synthesis*. Information, 2025. **16**(10): p. 874.
43. Martínez-Aguilar, S., et al., *Systematic Literature Review on Effort Estimation by Software Development Life Cycle Phases*. IEEE Access, 2025.
44. Khan, M.A., et al. *ML-Based Test Case Prioritization: A Research and Production Perspective in CI Environments*. in *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 2025. IEEE.
45. Ali, M., et al., *Generative adversarial networks (GANs) for medical image processing: recent advancements*. Archives of Computational Methods in Engineering, 2025. **32**(2): p. 1185–1198.
46. ŞAHiN, E., N.N. Arslan, and D. Özdemir, *Unlocking the black box: an in-depth review on interpretability, explainability, and reliability in deep learning*. Neural Computing and Applications, 2025. **37**(2): p. 859–965.
47. Feldt, R., F.G. de Oliveira Neto, and R. Torkar. *Ways of applying artificial intelligence in software engineering*. in *Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. 2018.
48. Farabi, S., *AI-driven predictive maintenance model for DWDM systems to enhance fiber network uptime in underserved US regions*. Preprints. https://doi.org/10.20944/preprints202506, 2025. **1152**: p. v1.
49. Hirth, M., et al., *Unsupervised deep learning for anomaly detection in automotive trucks: A survey*. ACM Computing Surveys, 2025. **58**(5): p. 1–35.
50. Priyanka, K., et al. *Self-Healing Data Pipelines: Reinforcement Learning for Real-Time Fault Detection and Autonomous Recovery*. in *2025 International Conference on Metaverse and Current Trends in Computing (ICMCTC)*. 2025. IEEE.
51. Laichapis, M., et al., *Financial Feasibility of Developing Sustained-Release Incrementally Modified Drugs in Thailand's Pharmaceutical Industry: Mixed Methods Study*. JMIRx Med, 2025. **6**(1): p. e65978.
52. Chountalas, P.T. and A.G. Lagodimos, *Integrated management systems: a content and bibliometric analysis*. The TQM Journal, 2025. **37**(7): p. 1827–1873.