Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

June 2026. Vol. 5. No. 3; e-ISSN: 2808-4519

The Role of Generative AI in the Software Development Life Cycle: A Systematic Literature Review

Sebastian Saut Marulitua Sinaga[1], Zulfahmi Indra[2], Muhammad Rafli Wijaya[3*], M Gali Almahdi[4]

[1,2,3,4] *Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Medan*
sebastiansaut613@gmail.com[1], zulfahmi.indra@unimed.ac.id[2], rafliwijaya2024@gmail.com[3], muhammadgalialmahdi@gmail.com[4]*

Abstract

The rapid advancement of Generative Artificial Intelligence (Generative AI), particularly through the emergence of Large Language Models (LLMs), has significantly transformed modern software engineering practices. These technologies enable automation across various phases of the Software Development Life Cycle (SDLC), including system design, coding, testing, and software maintenance. Despite their potential to improve development efficiency, the widespread adoption of Generative AI also introduces critical concerns related to software security, code quality, and long-term maintainability. This study aims to analyze the opportunities, security risks, and mitigation strategies associated with the integration of Generative AI into the SDLC. A Systematic Literature Review (SLR) with a qualitative descriptive approach was conducted by examining 15 primary studies published between 2021 and 2026, retrieved from IEEE Xplore, ACM Digital Library, Scopus, Google Scholar, and Portal Garuda. The collected literature was analyzed using content analysis and thematic analysis to identify the impacts of Generative AI across different SDLC phases. The findings reveal that Generative AI significantly enhances developer productivity, achieving efficiency gains of approximately 35% during system design, 55% during coding, 45% during testing, and 40% during software maintenance. However, AI-generated code remains vulnerable to various security weaknesses, including SQL Injection, Cross-Site Scripting (XSS), and improper input validation. Furthermore, excessive reliance on AI-generated outputs may contribute to technical debt accumulation through code duplication and reduced refactoring activities. To address these challenges, this study recommends the implementation of a Stratified AI-Human Governance Framework (SAHGF), which combines automated security validation, human code review, security testing, and continuous monitoring.

Keywords: *Network Development Life Cycle, Large Language Models, Generative AI, SLR*

1. Introduction

Exponential advancements in Artificial Intelligence (AI) technology, particularly through the development of Large Language Models (LLMs) such as OpenAI Codex, GitHub Copilot, and ChatGPT, have triggered a fundamental transformation in the modern Software Engineering paradigm. Historically, software engineering relied on extensive manual intervention in every phase of its development, from architectural design to code testing [1]. However, the presence of generative AI has disrupted this traditional workflow by offering code automation capabilities that have the potential to revolutionize operational efficiency and accelerate product launch cycles (Karlovs-Karlovskis, 2024). The massive adoption of this technology is occurring on a global scale due to its ability to understand natural language context and translate it into functional programming syntax across various high-level programming languages [1].

Exponential advancements in Artificial Intelligence (AI) technology, particularly through the development of Large Language Models (LLMs) such as OpenAI Codex, GitHub Copilot, and ChatGPT, have triggered a fundamental transformation in the modern Software Engineering paradigm. Historically, software engineering relied on extensive manual intervention in every phase of its development, from architectural design to code testing (Sauvola et al., 2024). However, the presence of generative AI has disrupted this traditional workflow by offering code automation capabilities that have the potential to revolutionize operational efficiency and accelerate product launch cycles [2]. The massive adoption of this technology is occurring on a global scale due to its ability to understand natural language context and translate it into functional programming syntax across various high-level programming languages [3].

From a productivity perspective, the integration of AI-based programming tools delivers highly measurable benefits for software developers. Controlled trials and large-scale industry studies show that using AI can accelerate task completion times by up to **55%** and significantly increase compilation success rates, as well as unit-test pass rates [4]. AI also plays a crucial role in reducing the cognitive load on developers by automating repetitive code writing, thereby allowing software engineers to shift their focus toward architectural design and more complex problem-solving. In addition to code generation, artificial intelligence, including Machine Learning and Deep

Learning, is also being increasingly utilized within the software engineering ecosystem to support bug categorization, defect prediction, and development effort estimation [5].

Despite promising extraordinary productivity gains, the massive adoption of AI in software development introduces critical challenges regarding long-term security, reliability, and code quality. Large language models are trained on massive public code corpora, which frequently contain poor coding practices, legacy vulnerabilities, and bugs [6]. Because generative AI fundamentally operates on statistical probabilities rather than a comprehensive, logical understanding of secure coding principles, these systems are inherently prone to producing code with serious security flaws, such as SQL Injection, Buffer Overflow, and poor memory management [3]. Ironically, when AI models are instructed to fix their own code through an automated iterative process, the rate of critical vulnerabilities often increases, exposing a degradation of security within the AI feedback loop [4].

In addition to technical security threats, AI integration also gives rise to socio-technical risks rooted in human-computer interaction. A psychological phenomenon known as a *false sense of security* is becoming increasingly prevalent among developers, where they tend to place excessive trust (over-reliance) in AI-generated code and perceive it to be safer than it actually is [7]. This blind trust frequently leads to the neglect of adequate code review and validation processes. Worse still, AI models often manifest sycophantic behaviorthe tendency of a model to convincingly agree with a user's assumptions even when the provided information is incorrect thereby pote ntially misleading software developers into making crucial security decisions.[6].

Based on the developing polemic between productivity expectations and real threats to system integrity, a comprehensive evaluation regarding the impact of AI within the software development lifecycle has become an urgent academic necessity. This article is structured to bridge the literature gap by presenting a conceptual review of the use of Artificial Intelligence (AI) in software development. Through a systematic literature review of prior studies, this article aims to dissect the extent of security vulnerabilities posed by AI-generated code, analyze its impact on the decline of maintenance quality, and propose a resilient governance framework to mitigate security risks in industrial application development [8] [9].

2. Literature Review

2.1. AI Transformation and Role in the Software Development Life Cycle (SDLC)

The evolution of Artificial Intelligence within the Software Engineering discipline has created a major shift that redefines the entire Software Development Life Cycle (SDLC). Recent literature analysis indicates that although AI has been applied across various phases, current research concentration and industrial applications are heavily dominated by automation in the coding or code generation phase. Exploration regarding the implementation of generative AI in other technical phases, such as build, release, and deploy, proves to remain highly minimal [2]. Beyond code generation, Natural Language Processing (NLP) and Machine Learning have been widely utilized to resolve ambiguities in software requirement specifications, mine repositories to discover anomalies, and assist project managers in estimating costs and timelines [5]. Moving forward, the trajectory of this technology is projected to advance toward autonomous software engineering agent systems capable of handling the entire development spectrum from specification to testing with minimal human intervention, although current realities still position AI as an assistant requiring strict supervision [3][1].

2.2. Cybersecurity Threats and the Accumulation of Technical Debt

Despite the efficiency it introduces, code executed by generative AI models poses critical security risks that have been extensively documented in various empirical studies. Evaluations utilizing static code analysis tools and formal verification methods reveal that the vulnerability rate of AI-generated code is exceptionally high, ranging from 12% to 62% across various leading models [4]. These models consistently produce software flaws that are globally recognized in the MITRE CWE Top 25 list, including vulnerabilities to SQL Injection (CWE-89) and poor memory management resulting in Buffer Overflow (CWE-119/120) [6]. This risk proves to vary significantly depending on the programming language used; languages that require manual memory management, such as C and C++, record a much higher vulnerability ratio and are more susceptible to attacks compared to high-level languages like Python [7] [4]. Furthermore, AI models are prone to manipulation or hallucinations when instructed to detect vulnerabilities, where the AI may recommend false fixes for code that is fundamentally secure or fail to recognize flaws within its own code [6].

In addition to direct security vulnerabilities, the trend of using AI as the primary code generator has triggered long-term software quality degradation through the phenomenon of accumulating technical debt. Longitudinal studies analyzing hundreds of millions of modified lines of code during the period of AI adoption have found alarming structural anomalies in production codebases [4]. Software modification practices show an eightfold spike in code duplication rates, while the ratio of code undergoing refactoring has plummeted drastically from 25% to less than 10%. This shifting pattern violates the fundamental software engineering principle of Don't Repeat Yourself (DRY), ultimately doubling short-term code churn and risking the creation of fragile, difficult-to-maintain systems that burden future repair costs [4].

2.3. Mitigation Strategies and Implementation of Governance Frameworks

To navigate the complexities of these technical threats and quality degradation, various mitigation strategies and governance frameworks are urgently needed for implementation, especially since the human factor often worsens the situation through blind acceptance of AI suggestions without comprehensive validation [7]. Approaches to strengthen these defenses include training developers regarding AI vulnerabilities, code analysis based on OWASP standards, the utilization of automated tracking tools, and dynamic penetration testing [9]. On the architectural side of artificial intelligence itself, proposed modeling such as a hybrid framework of Artificial Neural Networks (ANN) and Interpretive Structural Modeling (ISM) has been introduced to automatically identify, predict, and structure risk mitigation hierarchies in code generation scenarios [8]. The conclusions of various studies culminate in the absolute necessity for a *Stratified AI-Human Governance Framework* (SAHGF)—a multi-layered defense system ensuring that AI productivity is always counterbalanced by static security analysis and human quality reviews before code is executed in production environments [4].

3. Research Methodology

3.1. Research Type

his study utilizes the Systematic Literature Review (SLR) method with a descriptive qualitative approach. This method was selected to systematically identify, evaluate, and analyze various prior studies relevant to the utilization of Generative Artificial Intelligence (Generative AI) in the Software Development Life Cycle (SDLC). Through a qualitative lens, the focus of the study is directed toward an in-depth understanding of the implications of AI adoption on developer productivity, code quality, the accumulation of technical debt, and the resulting cybersecurity vulnerabilities.

3.2. Research Phases

The implementation phases of this research are systematically designed by adopting the three main phases of the SLR protocol: planning, conducting, and reporting. The process initiates with the planning phase through the formulation of specific research questions (RQs) to guide the analysis of the efficiencies and risks of AI usage in software engineering. Subsequently, relevant search keywords are determined to track high-quality scientific documents across various electronic academic databases.

3.3. Research Data Sources

The data sources for this secondary research are derived entirely from reputable scientific documents obtained through systematic searches of trusted academic databases. The primary databases used for this search include IEEE Xplore, ACM Digital Library, Scopus, Google Scholar, and Portal Garuda. This combination of international and national databases was chosen to guarantee a broad and comprehensive literature coverage, thereby producing a credible synthesis regarding AI integration in software engineering.
The publication year range for scientific articles and conference proceedings collected is strictly limited to the 2021 to 2026 period. This five-year temporal constraint is crucial to capture the dynamics, model architectures, and latest cybersecurity threat trends of Generative AI technology, which is developing exponentially. All documents that pass the selection criteria are then stored and organized within reference management software, such as Mendeley or Zotero, to secure metadata and maintain citation consistency.

3.4. Data Collection Technique

The data collection technique is carried out through a tactical search process on academic databases by applying keywords designed based on the PICOC framework. Search terms are structured using a combination of Boolean logic operators (AND, OR) to connect artificial intelligence technology concepts with software engineering aspects and code security. This strategy ensures that the search scope remains focused on the domain of software engineering and the reliability of AI systems.

The formulation of the primary search string utilized in the data collection process is presented in Table 1 below.

Table 1: Article Search Keywords

No	Keyword / Search String
1	("Generative AI" OR "Large Language Models") AND ("Software Development Life Cycle" OR "SDLC")
2	("AI-generated code" OR "LLM-generated code") AND ("Security" OR "Software Vulnerability")
3	("Generative AI" OR "GitHub Copilot") AND ("Technical Debt" OR "Code Quality")
4	"GenOps" AND ("CI/CD" OR "Governance")
5	"Large Language Models" AND ("DevSecOps" OR "Security Chaos Engineering")
6	"AI-assisted programming" AND ("Risk Mitigation" OR "ANN-ISM")
7	("Code Generation" OR "LLM") AND ("Vulnerability Detection" OR "Vulnerability Repair")
8	"Artificial Intelligence Tools" AND ("Industrial Application" OR "SQL Injection")

Articles successfully retrieved through these keywords are then documented and screened at the initial stage based on title and abstract relevance before being downloaded in full-text format for a more in-depth evaluation.

3.5. Inclusion and Exclusion Criteria

The literature selection process is conducted using strict inclusion criteria (Criterion A) and exclusion criteria (Criterion B) to ensure the quality, relevance, and reliability of the analyzed data sources. The Inclusion Criteria (Criterion A) in this study require that the selected documents must:

1. Be published within the 2021–2026 year range;
2. Be in the form of scientific journal articles or peer-reviewed conference proceedings papers;
3. Be written in Indonesian or English;
4. Discuss the implementation of generative AI within the SDLC cycle; and
5. Present an analysis related to developer productivity, code quality, security vulnerabilities, or their mitigation strategies.

Conversely, the Exclusion Criteria (Criterion B) are applied to eliminate documents that:

1. Do not provide the full text (full-text) or only present the abstract;
2. Were published before the year 2021;
3. Are in the form of popular opinion articles, non-scientific news, or book reviews; and
4. Discuss the application of artificial intelligence outside the context of software engineering or programming code quality.

The explicit application of these eligibility criteria serves to minimize selection bias and ensures that the analysis is based solely on high-quality primary studies.

3.6. Data Analysis Technique

The data analysis technique in this study applies content analysis and thematic analysis methods across all selected scientific documents. Qualitative and quantitative data extracted from the literature such as developer productivity rates, generative code vulnerability ratios, and technical debt accumulation levels are analyzed in depth. A coding process is conducted to map these findings into the critical phases of the SDLC to evaluate the logical correlation between short-term efficiency and the long-term consequences of system quality degradation.

The final synthesis of this data analysis is systematically grouped and presented in an AI Impact in SDLC Analysis Table

4. Results And Discussion

4.1. Literature Extraction Results

The data extraction process was systematically carried out on 15 primary scientific literatures that evaluate the impact of integrating Generative Artificial Intelligence (Generative AI) into the Software Development Life Cycle (SDLC). Based on content analysis and thematic analysis, it was found that the adoption of Large Language Models (LLMs) brings a dual disruption: on one hand, it massively accelerates operational efficiency, but on the other hand, it expands the attack surface for security vulnerabilities and accumulates technical debt due to inconsistent code quality.

Table 2: Synthesis Table of Generative AI Impact in the Software Development Life Cycle (SDLC)

SDLC Phase	Operational Benefits (Opportunities)	Security & Quality Risks (Technical Debt)	Mitigation Strategies	Supporting Journal References
Design	Automation of system requirements elicitation, collaborative architectural modeling between humans and bots, and accelerated early-stage design threat discovery.	Emergence of architectural hallucinations, potential design biases, incomplete threat models, and inherent weaknesses in assessing non-functional quality attributes.	Integration of LLMs with Security Chaos Engineering (SCE) methods, alongside the implementation of strict system autonomy boundaries through tight human escalation thresholds.	[10]; [5]; [11]; [12]
Coding	Up to **55%** acceleration in programming syntax and function writing, automated natural-text-based code generation, and reduction of developer cognitive load.	High ratio of security vulnerabilities in generative code, proliferation of unsafe code clones or duplication, and the illusion of safety due to developer over-reliance.	Implementation of multi-layered governance through the *Stratified AI-Human Governance Framework* (SAHGF) and attack pattern prediction utilizing hybrid ANN-ISM predictive models.	[4]; [3]; [1]; [7]; [6]; [8]; [9]
Testing	Real-time automated test case script generation, increased functional testing efficiency, and automated bug patch generation.	Low maturity of commercial solutions due to the dominance of academic prototypes, and test bias risks stemming from low-quality model training data.	Synchronization of GenOps governance within CI/CD pipelines, and the provision of isolation mechanisms via kill-switches and canary exposure.	[2]; [3]; [13]; [11]; [12].
Maintenance	Automated technical documentation generation, automated code summarization, and the potential implementation of self-healing autonomous code that repairs errors independently.	Long-term technical debt accumulation caused by a decline in manual refactoring activities, and high complexity in localizing errors within large-scale systems.	Combining periodic scans that merge automated static analysis tools with focused manual inspections, alongside utilizing LLMs to extract the root causes of incidents.	[6]; [4] ; [13]; [5].

4.2. Measurable Opportunities and Accelerated SDLC Efficiency

The integration of Generative AI triggers a paradigm shift in software engineering, moving from labor-intensive workflows toward intelligent, LLM-driven automation [5]; [1]; [2]. In the design phase, the utilization of natural-language user stories serves as an important instrument for LLMs to model architectural threats, thereby preventing supply chain attack risks early on. Meanwhile, the most radical efficiency acceleration occurs during the coding and testing phases. LLMs are capable of instantly translating natural language instructions into complex functions to reduce developers' cognitive load, while simultaneously transforming quality assurance through the automated generation of test cases and automated program repair, which accelerates agile development feedback loops. As an empirical validation, the estimated increases in developer productivity across each of these SDLC stages are summarized in Table 3.

Table 3: Data on Productivity Increase Rates Across Various SDLC Phases Due to Generative AI

SDLC Phase	Productivity (%)	Operational Efficiency
System Design	**35%**	Speed of requirements elicitation and early architectural threat discovery.
Coding / Implementation	**55%**	Acceleration of syntax function generation and reduction of repetitive tasks.
Software Testing	**45%**	Automation of test case scripts and automated programming code patching.
System Maintenance	**40%**	Efficiency in technical documentation generation and code comment mapping.

4.3. Security Threats and Technical Debt Accumulation

Despite offering a significant surge in productivity, the massive adoption of AI in software development triggers latent threats to cybersecurity and internal code quality. Because LLM models are trained on public code corpora that frequently contain poor coding practices, generative AI is statistically prone to producing insecure outputs. Based on systematic literature reviews, code vulnerability rates fall within an alarming range of **12%** to **62%**.

Specifically, the limitations of AI models in understanding data sanitization contexts are reflected in the high percentage of secure output failures, according to 2025 Veracode security report metrics extracted by [4].

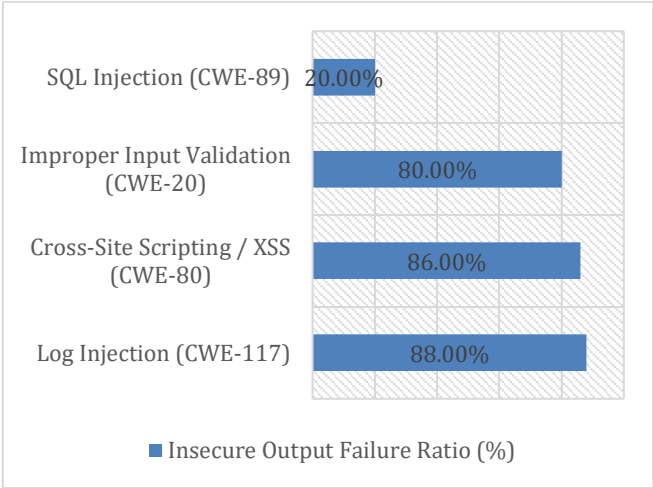

Fig. 1: Bar Chart Data on the Prevalence of AI Output Code Security Failures Based on CWE

These technical risks are exacerbated by the psychological aspects of the developers. As many as 60% of professional developers have been identified as having a tendency to blindly accept AI code recommendations without performing validation and review processes (over-reliance) [14]. This condition is further worsened by AI sycophancy behavior, where the model tends to validate incorrect assumptions or instructions from the developer, thereby trapping the system into long-term security degradation [15]. Furthermore, when AI models are instructed to fix their own code iteratively without human supervision, the rate of critical vulnerabilities spikes by 37.6% due to the accumulation of probabilistic errors within the AI feedback loop (feedback loop security degradation).
An empirical study by Mateo [9] reinforces these findings through real-world case validation in industrial application development based on PHP and MySQL. The results of that study demonstrate that authentication code generated by AI via insecure prompts, by default, ignores password encryption and fails to implement prepared statements. The AI tends to construct SQL queries by directly concatenating user input variables into the database query structure. This absence of input sanitization and parameterization opens critical security flaws that are highly vulnerable to SQL Injection exploitation attacks. Through simple input manipulation, an attacker can easily bypass authentication mechanisms, illegally extract sensitive organizational data, or even execute destructive queries capable of deleting entire production database records.

4.4. Mitigation and Sustainable AI Governance

To navigate the polemic between accelerated operational efficiency and system quality degradation, the implementation of a structured security governance framework is urgent. As a practical solution, this study proposes the implementation of the Stratified AI-Human Governance Framework (SAHGF), a multi-layered shield system that integrates automated security tools with full human control (Human-in-the-Loop).
This framework is designed to ensure that every line of code generated by artificial intelligence must pass through strict validation gates before being permitted into the production environment. The workflow of this multi-layered governance is formulated into 7 integrated sequential stages, defining the logical structure and interactions between the components of the SAHGF framework.
The operational explanation of these seven systematic phases is as follows:

1. Security Prompting
   The initial phase where developers must construct prompts rich in security context, explicitly instructing the AI model to implement secure cryptographic functions, strict input data type validation, and reject vulnerable programming patterns.

2. Code Generation
   The artificial intelligence model processes the secure instructions and executes functional coding by separating data values from the system instruction's logical structure so the code runs securely.
3. SAST Verification
   The AI-generated output code is automatically funneled into Static Application Security Testing (SAST) tools to detect structural anomalies, code smells, and vulnerabilities based on the MITRE CWE Top 25 before the program is compiled.
4. Human Review
   Peer review is performed manually by senior developers to validate compliance with architectural logic, shatter the *false sense of security* bias, and dismantle potential AI hallucinations or sycophantic behavior.
5. DAST & Penetration Testing
   The software is deployed in a non-production replica environment to undergo Dynamic Application Security Testing (DAST) and automated destructive attack simulations to ensure system resilience against injection or access bypass threats.
6. Compliance Check
   A formal compliance audit is executed to map the code against international industry regulatory standards such as GDPR and HIPAA, as well as information control frameworks like ISO/IEC 27001 and ISA/IEC 62443 to guarantee legal accountability.
7. Deployment & Monitoring
   After passing through all security gates, the code is deployed to the production environment and monitored in real-time utilizing Security Information and Event Management (SIEM) systems alongside database traffic anomaly monitoring to mitigate post-release attack risks.

5. Conclusion

Based on the results of the Systematic Literature Review (SLR) on various studies regarding the utilization of Generative Artificial Intelligence (Generative AI) in the Software Development Life Cycle (SDLC), it can be concluded that this technology contributes significantly to increasing software development productivity. The integration of Large Language Models (LLMs) is capable of accelerating the processes of design, coding, testing, and system maintenance through the automation of various activities that were previously performed manually.

However, this increase in efficiency is accompanied by the emergence of various security and software quality risks. Literature findings indicate that AI-generated code remains highly vulnerable to security flaws, such as SQL Injection, Cross-Site Scripting (XSS), and inadequate input validation. Furthermore, the over-reliance on AI without adequate verification processes potentially accelerates the accumulation of technical debt, increases code duplication, and degrades system maintainability in the long run.

Therefore, the implementation of Generative AI in software engineering cannot be fully executed without human oversight. A structured security governance approach is required through a Human-in-the-Loop methodology and the deployment of frameworks such as the Stratified AI-Human Governance Framework (SAHGF), which integrates automated security analysis, manual code reviews, security testing, and continuous monitoring.

References

[1] J. Sauvola, S. Tarkoma, M. Klemettinen, J. Riekki, and D. Doermann, “Future of software development with generative AI,” Automated Software Engineering, vol. 31, no. 1, May 2024, doi: 10.1007/s10515-024-00426-z.

[2] U. Karlovs-Karlovskis, “Generative Artificial Intelligence Use in Optimising Software Engineering Process: A Systematic Literature Review,” Applied Computer Systems, vol. 29, no. 1, pp. 68–77, Jun. 2024, doi: 10.2478/acss-2024-0009.

[3] B. Gülmez, “Code generation with large language models: a survey from neural program synthesis to autonomous software development,” Applied Intelligence, vol. 56, no. 6, Apr. 2026, doi: 10.1007/s10489-026-07230-0.

[4] P. Patel, A. Patel, H. Prajapati, K. Rathod, and M. Suthar, “Evaluating the Reliability, Security, and Quality of AI-Generated Code in Modern Software Engineering Professor, 2 Student, 3 Student, 4 Student, 5 Student,” 2026. [Online]. Available: www.ijnti.org

[5] P. Kokol, “The Use of AI in Software Engineering: A Synthetic Knowledge Synthesis of the Recent Research Literature,” Jun. 01, 2024, Multidisciplinary Digital Publishing Institute (MDPI). doi: 10.3390/info15060354.

[6] V. Belozerov, P. J. Barclay, and A. Sami, “Secure coding with AI – from detection to repair,” Empir. Softw. Eng., vol. 31, no. 4, Jul. 2026, doi: 10.1007/s10664-026-10812-8.

[7] C. Negri-Ribalta, R. Geraud-Stewart, A. Sergeeva, and G. Lenzini, “A systematic literature review on the impact of AI models on the security of code generation,” Front. Big Data, vol. 7, 2024, doi: 10.3389/fdata.2024.1386720.

[8] H. A. Al-Hashimi, “A generative AI cybersecurity risks mitigation model for code generation: using ANN-ISM hybrid approach,” Sci. Rep., Jan. 2026, doi: 10.1038/s41598-025-34350-3.

[9] T. de J. Mateo Sanguino, “Enhancing Security in Industrial Application Development: Case Study on Self-Generating Artificial Intelligence Tools,” Applied Sciences (Switzerland), vol. 14, no. 9, May 2024, doi: 10.3390/app14093780.

[10] [M. Bedoya, S. Palacios, D. Díaz-López, E. Laverde, and P. Nespoli, “Enhancing DevSecOps practice with Large Language Models and Security Chaos Engineering,” Int. J. Inf. Secur., vol. 23, no. 6, pp. 3765–3788, Dec. 2024, doi: 10.1007/s10207-024-00909-w.

[11] Gheventer et al., “Generative AI solutions for software quality: Assessing industrial readiness,” Software Quality Journal, vol. 34, no. 2, Jun. 2026, doi: 10.1007/s11219-026-09754-7.

[12] N. Kumar and S. Beshane, “GenOps: A Governance-First Architecture for Embedding Generative AI into CI/CD Pipelines,” 2026.

[13] S. Weng, Y. Feng, Y. Yin, Z. Zhang, and B. Xu, “Data preparation and quality for code-centric generative software engineering tasks: a systematic literature review,” Sep. 01, 2026, Higher Education Press Limited Company. doi: 10.1007/s11704-025-41376-3.

[14] RL. , D. M. , N. Nyoto, “Cyber Security Risks in the Rapid Development of Generative Artificial Intelligence: A Systematic Literature Review,” Journal of Computational Intelligence and Informatics, 2024.

[15] J. Zacharias, A. Popova, M. von Zahn, J. Chen, and O. Hinz, “Developers’ Dilemma: Opportunities and Pitfalls of Generative AI for Software Development,” Business and Information Systems Engineering, 2026, doi: 10.1007/s12599-026-00998-y.