International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

Assessing the Impact of Generative AI on Code Quality and Software Maintainability

Dr. Ashish Jolly
Associate Professor, Department of Computer Science,
Govt. PG College, Ambala Cantt

Abstract
Generative artificial intelligence (GenAI) tools have radically transformed the software development landscape because they may assist in the code creation, refactoring, and debugging. Although these tools are gaining popularity on open-source and non-open-source projects, there is a need to consider the impact they have on the quality and maintainability of the code. The effect of GenAI on the software quality is analyzed in this review study with the focus on the long-term maintainability. This paper evaluates the pros and cons of AI-generated code versus human-written code by synthesising past studies and the experimental studies, highlighting the trade-offs between short-term productivity and long-term code health. As it suggests the recommended practices in the responsible integration of the products of GenAI, it sheds light on such crucial concerns as security vulnerabilities, technical debt, and lack of understanding. It is stated in the conclusion of the paper that more research
is desired in order to develop frameworks and principles of maximising AI-assisted development.

Keywords: Generative AI, software maintenance, code quality, large language model,

I. INTRODUCTION

Generative artificial intelligence (AI) has played a disruptive role in the recent years in software engineering. Large language models (LLMs) trained on large natural language corpora and source code can be used to generate and suggest code snippets, provide support during all phases of the software development lifecycle through generative AI technologies. Since they are able to accelerate the coding workflow, reduce repetition, and provide context-sensitive suggestions, GitHub Copilot, ChatGPT, Tabnine, and CodeWhisperer are gaining popularity in the industry and academia [1].

These advances cast grave doubts on their impact on the quality of the software and the sustainability of its maintenance despite the potential huge productivity benefits. To achieve reliability and efficiency of systems, the quality of the code used as a measure of how accurately, readably, efficiently and robust the software meets the requirements is essential [2,3]. Software maintainability, or ease with which code can be understood, modified and extended with time, is also a very crucial factor, especially in large and dynamic code bases. Increased errors, increased costs in maintaining it, and reduced ability to adapt to changing needs can be brought by low maintainability or code quality.

The use of generative AI in programming processes can be characterized as a paradigm change. Although automated recommendations can perform certain human cognitive functions, this comes with the risk of inconsistency, the existence of underlying technical debt, and unclear logic[4,5]. Early empirical studies and reports of the industry have indicated mixed results; some have indicated that everyday tasks become more effective when aided by AI, others have indicated that there is complexity unexpected or that there are a few flaws in AI-assisted code. Although the topic has become more popular, not all the effects of generative AI on code quality and maintainability across different contexts are yet summarized.

This review paper aims to fill that gap by evaluating important discoveries of the past, discussing new advances in applying generative AI to the programming of software. The paper examines the possible benefits of generative AI and the challenges and limitations that must be addressed to ensure its responsible and successful use through a systematic review of the empirical evidence and subjective views of professionals.

A. Objectives

The key objectives of this review paper are as follows:

- To examine how AI-based coding helpers are currently used across the software development lifecycle to explore the generative AI application in modern software development.
- To evaluate the impact of generative AI on code quality, it is necessary to consider such important attributes as accuracy, readability, consistency, efficiency, and adherence to the coding standards.
- To study the impact of generative AI on software maintainability, considering such aspects as technical debt, modularity, documentation quality, and ease of further modification.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

- To provide a review of existing research tendencies and gaps by integrating information and existing empirical studies on the topic through industry reports.
- To emphasize the deficiencies of the tools and studies which have now been made, where the further research is necessary, most particularly in the case of large-scale software systems and long-term maintainability.
- To offer the best practices and the possible research areas concerning the introduction of the generative AI into the processes of software engineering and maintain the high quality of the code and the software development life cycle.

II. RELATED WORK

A. Evolution of AI in Software Engineering

The last decade has seen a significant transformation in the application of artificial intelligence (AI) in software engineering (SE): instead of the traditional rule-based system, complicated generative models are used and they actively engage in the coding and creation of systems. To aid design and testing processes, automated planning, model-driven engineering and expert systems were the primary concerns of early AI research in SE. However, with the development of machine learning and natural language processing, AI became more active in such areas as automatic documentation, code generation, and debugging.

This evolution has involved the use of machine learning techniques as an input in the process of software engineering. The synthesis of the existing research[6] indicates that the systematic change in the concept of software building gradually occurred with the gradual incorporation of AI into the range of SE processes such as requirement analysis, coding, and quality assurance.

Generative AI was also introduced into integrated development environments (IDEs) (with the assistance of Copilot, initially enabled by the Codex model of OpenAI) to allow a developer to write code based on the contextual indicators and natural language prompts in response to preexisting code. This has developed a new paradigm where AI works alongside coding processes and significantly minimized the barriers to automating routine coding processes.

With AI in SE, the concept of autocompletion developed into context-based recommendations potentially able to create entire snippets and test cases, with similar developments being made independent discoveries such as the AI code completion system Tabnine, which expanded on the foundation of Codota. The scalability of these technologies to other languages and IDEs became possible due to the introduction of deep learning architectures, which illustrate an even greater shift towards intelligence-augmented programming environments. Saravanan et al. [7] state that the use of generative AI models to automate tasks hitherto performed by human developers has begun to affect the basic development lifecycle, resulting in higher productivity, and emerging research issues. Recent literature also suggests a broader change in the practice of software engineering, highlighting the way this change is altering the practitioner and teaching needs. Rather than writing all the code by hand, developers have to learn how to effectively prompt and assess AI outputs.

B. Code Quality and AI-Generated Code

With the growing deployment of generative AI models, and in particular the large language models (LLMs), in software development, recent research has put greater emphasis on measuring the quality of code output of these models. Empirical studies also show that AI generated code often passes basic functionality tests and is syntactically correct, specifically to well-defined and commonly-used programming tasks [8]. Even though AI systems are more likely to generate less intricate and repetitive structures, comparative analyses of human-written and AI-generated code also reveal that the latter systems are more prone to breaching best practices, redundant logic, and adherent patterns[9]. Also, it has been demonstrated that functional correctness and the general quality of the code are not closely related, and passing tests alone does not guarantee the security, readability, and maintainability of the code. Table 1 presents the related work with representative studies.

Table I: Related work

Theme	Key Points	Representative Studies / Examples
Evolution of AI in Software Engineering	• AI has evolved from simple completion to generating full code from natural language. • Large language models (LLMs) like transformers improved code generation. • Reviews highlight broader impacts on productivity and workflows.	• Vaswani et al., 2017 (transformers) [10] • GitHub Copilot (OpenAI Codex) • Brunt et al., 2022 (AI in coding workflows) [11]
Surveys & Systematic Reviews on Generative AI in SE	• Systematic literature reviews show broad research interest in code generation	• Karlovs-Karlovskis et al. (2024) – Generative AI in SE (117 studies)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

Theme	Key Points	Representative Studies / Examples
	and related SE tasks since ~2020. • Reviews highlight efficiency gains alongside quality challenges. • Other reviews examine AI-assisted code generation and code review.	(Paradigm) [12] • Wang – Review of AI-assisted code generation & review (Darcy & Roy Press) [13]
Code Quality and AI-Generated Code	• AI can generate correct code but often lacks best practices for readability and maintainability. • Empirical evaluations compare quality across tools (Copilot, CodeWhisperer, ChatGPT). • Reviews raise concerns about security vulnerabilities in AI code.	• Yetiştiren et al. (2023) – empirical tool comparison [14]
AI-Assisted Code Review & Quality Tools	• AI tools automate code review, static analysis, and contextual feedback, improving productivity and catching issues. • Automated review research focuses on practical adoption and effects on review time/quality.	• Jangam & Karri – review of AI code review tools [15] • Cihan et al. (2024) – study of automated code review in practice [16]
Software Maintainability	• AI-generated code may increase technical debt when used without design planning. • Lack of documentation and contextual understanding complicates long-term maintenance. • Surveys and practitioner reports often note repetitive or "vibe coding" patterns that hurt maintainability.	• Dey et al., 2022 [17]

C. Software Maintainability

Recent studies have been interested in the effect of generative AI on software maintainability, investigating the effect of AI-generated code on refactoring methods, long-term software development, and the ability of subsequent developers to read and modify code. However, empirical studies comparing code generated by models like GPT-4 to code written by humans indicate that even when functional correctness is achieved at runtime, code by models has more cyclomatic complexity and repetition patterns, which have adverse effects on maintainability indicators and are likely to increase the process of technical debt when it is not properly mitigated, even though insights into these trends are scarce. This implies that the influence of AI can be illusory and soft. These results are also in line with the accompanying studies on the downstream impact of AI support on code evolution.

As well, large-scale analysis of properties of structural codes reveals that as much as AI can be useful in localised refactoring and the removal of some code smells, it may also lead to maintainability issues such as redundant logic or non-idiomatic patterns in source files and build scripts. These are early findings that have so far focused on tool enhancement, as opposed to truly autonomous maintenance, but studies on AI-based refactoring and methods that utilize advanced models such as Graph Neural Networks indicate that their complexity and coupling metrics are improving, the latter being significant measures of maintainable design. This challenge facing the practitioners is further emphasized by industry reports, which indicate that human control remains the only way to keep software healthy in the long run. The developers tend not to trust AI-generated code, and thus they have to do additional verification and remediation tasks to maintain it before integrating it. The literature in general suggests that despite the ability of generative AI to supporting maintainability with focused refactoring and documentation, its total effect on long term maintainability is compound and requires quality controls, integration processes, and complexity of the software system.

III. IMPACT OF GENERATIVE AI ON CODE QUALITY

A. Readability and Code Complexity.

This feature can also give less cognitive load on the developers and make the development faster when developing routine functionalities. However, some

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

studies demonstrate that the syntactically correct AI-generated code is often not well aware of the rest of the application context. As a result, such code may not necessarily comply with the rules of style, architectural principles, or with the names of the projects. This misalignment can result in poor code readability and make the codebase more difficult to understand, appraise and maintain by the members of the team. Since AI-generated logic is often difficult to interpret, developers are often forced to work harder to get the original productivity benefits [18].

B. Code Redundancy and Technical Debt.

Another meaningful implication that generative AI has on the quality of code is the propensity to introduce unnecessary coding. Rather than reusing an existing functionality, class, or module, AI models tend to generate duplicate blocks of code or logic. This behaviour may otherwise work well in small scale or far-flung implementations, but as redundancy accumulates with time, it becomes a nuisance with larger systems. Any change or problem patch may need to be made in multiple locations, which could introduce the risk of errors, inconsistency, and time-intensive debugging and refactoring[19]. The long-term quality of software can then be undermined by unchecked artificial intelligence generated redundancy.

C. Security Weaknesses

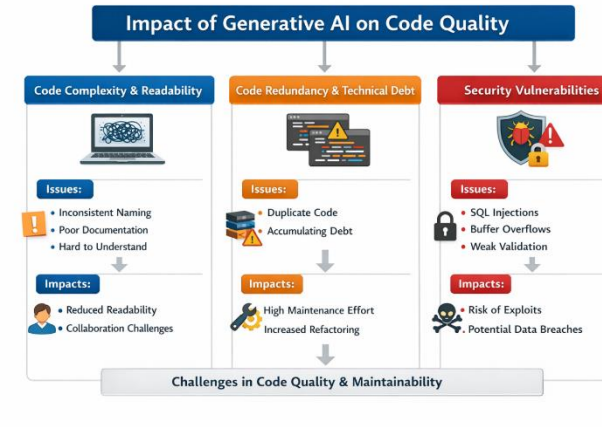

Fig 1: Impact of generative AO on code quality

Security is another problem with AI-generated programming that is severe. In turn, such security vulnerabilities as buffer overflows, SQL injection, and improper input validation can be accidentally presented by the solutions produced by AI [20]. According to Zhou et al. (2023), generative AI systems cannot identify and mitigate potential security risks because they are not based on the deep semantic or security study but generate patterns. This code can become the target of exploitation and data leakage when it is incorporated into production systems without due security checks which is a grave threat to the confidence of users and the reliability of software[21].

The primary directions in which generative AI has an influence on software creation are graphically outlined in Figure 1. Generative AI in Software Development seems to be the central force at the core, where three main areas of influence are created: Code Complexity and Readability, Code Redundancy and Technical Debt, and Security Vulnerabilities. In the former, AI-created code often replaces the monotonous coding tasks and boilerplate writing, however, it is accompanied by such issues as the inconsistencies in naming, poor documentation, and difficulties in understanding the logic[22]. All these issues complicate the reading process and may complicate the collaboration of the developers. In the second section, Code Redundancy & Technical Debt, the authors underline the fact that AI-generated solutions can lead to redundancy and the inadequate reuse of the already existing modules, which can become a technical debt. Lastly, the Security Vulnerabilities exhibit that AI-generated code, which is based on the pattern-based generation of publicly available datasets, will unintentionally incorporate vulnerabilities like SQL injections, buffer overflows, or poor validation principles. Such vulnerabilities increase the chances of exploitation and possible breach of data within the production settings. In general, the diagram highlights that although generative AI can be efficient and fast, it presents significant readability, maintainability, and security issues, and it is necessary to examine it with human attention to these issues.

IV. LONG-TERM SOFTWARE MAINTAINABILITY

A. Code Evolution and Refactoring.

Refactoring is necessary to maintain quality of code, readability, and stability of the system as software system grows in size and complexity. Generative AI tools are increasingly being used to automatically clean up code, enforce naming conventions, break down large functions, or remove dead or superfluous code by developers, as the use of these tools rapidly grows in popularity. But AI is often not capable of tackling tricky architectural problems, although it is very good at superficial refactoring. Complex tasks such as the rearrangement of modules to enhance cohesiveness, solving tight connectivity among modules, cross-cutting problems, and performance optimization continue to be a challenge to AI systems [23]. This limitation is due to the fact that the AI relies on pattern-based code generation and not on architectural analysis, and is not aware of the system design worldwide. Hence, small issues such as broken abstractions, ineffective logic, or absent

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

error handling, or performance degradation, can be accidentally added through AI-based refactoring.

Also, the history of the project is not typically contained in the suggestions of AI technologies. The evolution of the future can be more challenging due to inconsistent solutions based on the principles of the long-term designs or architectural patterns. The developer becomes overly reliant and overworked by the application developers [24]. Normal development work requires a cognitive load and significant manual labour, which can be greatly minimized with the assistance of generative AI. Nonetheless, overreliance on AI is very dangerous in the context of the software systems durability and maintainability. The excessive use of AI-generated solutions can lead to a gradual loss of the understanding of the codebase that developers needed to make prudent decisions in the course of system evolution, refactoring, or debugging. This addiction could have a number of consequences:

- **Weakness of skills:** Developers themselves might fail to internalise design patterns, best practices or architectural reasoning, which restricts their ability to evaluate and improve code independently.
- **Weaker critical thinking:** When recommendations given by AI are unquestioningly followed without careful analysis, then the syntactically correct code, which is architecturally unsound or insecure, can be approved.
- **Lack of knowledge in teams:** Over time, fewer individuals in a team might have a holistic understanding of the system which may result in bottlenecks when major refactoring or maintenance is required[25].

Moreover, AI-programmed code often does not contain a proper understanding of context. Though it can give out functionalities, it cannot fully understand business logic, system requirement, or interdependences between modules [26]. Such a restriction might result in stand-alone code that is not difficult to maintain over the long term, but has problems such as the introduction of small bugs, non-uniform behaviour, or poor integration with dynamically changing parts of the system. Organisations ought to address these risks equally, using AI tools to improve efficiency without losing the active involvement of people to review, validate and evolve codes [27]. This approach will enable long-term and sustainable software development by making sure that AI is not a replacement of human knowledge, but an aid to it.

V. CONCLUSION

Generative AI has much potential to transform software development processes by enhancing the productivity of developers, accelerating their work, and reducing the workload of repetitive and boilerplate development. The technologies can significantly shorten the development cycles and liberate engineers to focus on the higher level design and problem solving by providing automated code generation, real-time prototyping, and prototyping. The benefits of these will be higher productivity and faster delivery of software products in the nearest future.

Nevertheless, generative AI creates a complex and far-reaching impact on the quality of the code and the maintainability of a long-term programme. This can eventually result in the accumulation of technical debt, name inconsistency, logic duplication and lower readability. Moreover, the fact that the training data is published and can be found on the publicly accessible code repositories questions the possibility of introducing security vulnerabilities and outdated or unsafe coding practices, which, in turn, might compromise the reliability of the systems.

Companies should have a balanced and responsible integration of generative AI in software engineering to make effective use of this technology. This requires the implementation of strict human-in-the-loop code reviews, automated testing and tool set of the and-well-known static tests, and security-auditing to identify and mitigate vulnerabilities early in the development lifecycle. Specific rules and best practices must also be established to ensure that the code generated by AI is stable enough to meet the expectations of the team and the long-term goals of maintainability. Ultimately, generative AI should be regarded as another tool that empowers the human knowledge instead of replacing it. The potential to produce software systems that are reliable, secure and maintainable with the application of generative AI is high in case it is implemented cautiously and in the context of the close control on the opportunities to design a high-quality assurance process.

References

[1] Dhanore, Y. M. (2025). The Impact of Generative AI Tools in Open-Source Software Development.

[2] Nguyen-Duc, A., Cabrero-Daniel, B., Przybylek, A., Arora, C., Khanna, D., Herda, T., ... & Abrahamsson, P. (2025). Generative artificial intelligence for software engineering—A research agenda. Software: Practice and Experience, 55(11), 1806-1843.

[3] Alwageed, H. S., & Khan, R. A. (2025, June). The role of generative AI in strengthening secure software coding practices: A systematic perspective. In Proceedings of the 2025 29th International Conference on Evaluation and Assessment in Software Engineering Companion (pp. 136-141).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

[4] Malladi, N. V., & Reddy, S. (2025, October). Generative AI in Agile Software Development: A Comprehensive Survey. In 2025 7th International Conference on Innovative Data Communication Technologies and Application (ICIDCA) (pp. 1199-1208). IEEE.

[5] Santos, P. D. O., Figueiredo, A. C., Nuno Moura, P., Diirr, B., Alvim, A. C., & Santos, R. P. D. (2024, May). Impacts of the usage of generative artificial intelligence on software development process. In Proceedings of the 20th Brazilian Symposium on Information Systems (pp. 1-9).

[6] Kokol, P., Jernej, Z., Blažun, V. H., & Bojan, Ž. (2025). Evolutionary Game Theory Use in Healthcare: A Synthetic Knowledge
Synthesis. Information, 16(10), 874.

[7] Ghosh, S., & Wilson, K. (2025, October). Bias Is a Math Problem, AI Bias Is a Technical Problem: 10-Year Literature Review of AI/LLM Bias Research Reveals Narrow [Gender-Centric] Conceptions of 'Bias', and Academia-Industry Gap. In Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society (Vol. 8, No. 2, pp. 1091-1106).

[8] Horan, W. P., Moore, R. C., Belanger, H. G., & Harvey, P. D. (2024). Utilizing technology to enhance the ecological validity of cognitive and functional assessments in schizophrenia: an overview of the state-of-the-art. Schizophrenia bulletin open, 5(1), sgae025.

[9] Ambati, S. (2023). Security and Authenticity of AI-generated code (Doctoral dissertation, University of Saskatchewan).

[10] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[11] Brunt, B., & Smith, Y. Technology and Artificial Intelligence (AI) in Nursing.

[12] Pabreja, K., Verma, R., & Kumar, A. (2025). GenAI: a scientometric analysis of research trends using biblioshiny and VOSviewer. Discover Artificial Intelligence.

[13] Wong, M. F., Guo, S., Hang, C. N., Ho, S. W., & Tan, C. W. (2023). Natural language generation and understanding of big code for AI-assisted programming: A review. Entropy, 25(6), 888.

[14] Yetiştiren, B., Özsoy, I., Ayerdem, M., & Tüzün, E. (2023). Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. arXiv preprint arXiv:2304.10778.

[15] Jangam, S. K. (2025). Advancements in AI Coding Assistance Tools and Their Potential Impact on Collaborative Software Development. American International Journal of Computer Science and Technology, 7(2), 1-14.

[16] Cihan, U., Haratian, V., İçöz, A., Gül, M. K., Devran, Ö., Bayendur, E. F., ... & Tüzün, E. (2025, April). Automated code review in practice. In 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 425-436). IEEE.

[17] Dey, P. K., Malesios, C., De, D., Budhwar, P., Chowdhury, S., & Cheffi, W. (2022). Circular economy to enhance sustainability of small and medium sized enterprises. In Supply chain sustainability in small and medium sized enterprises (pp. 10-45). Routledge.

[18] Dhanore, Y. M. (2025). The Impact of Generative AI Tools in Open-Source Software Development.

[19] Al-Fawareh, H., & Al-Shdaifat, H. M. (2025). Investigating AI-Generated Code on the Impact on Software Efficiency Code Quality Factor. In Sustainable Data Management: Navigating Big Data, Communication Technology, and Business Digital Leadership. Volume 1 (pp. 89-97). Cham: Springer Nature Switzerland.

[20] Song, F., Agarwal, A., & Wen, W. (2024). The impact of generative AI on collaborative open-source software development: Evidence from GitHub Copilot. arXiv preprint arXiv:2410.02091.

[21] Daniotti, S., Wachs, J., Feng, X., & Neffke, F. (2026). Who is using AI to code? Global diffusion and impact of generative AI. Science, eadz9311.

[22] Fawareh, H., Al-Shdaifat, H. M., Al-Refai, M., Fawareh, F. A., & Khouj, M. (2024, December). Investigates the Impact of AI-generated Code Tools on Software Readability Code Quality Factor. In 2024 25th International Arab Conference on Information Technology (ACIT) (pp. 1-5). IEEE.

[23] Molnar, A. J., & Motogna, S. (2020). Longitudinal evaluation of open-source software maintainability. arXiv preprint arXiv:2003.00447.

[24] Bakota, T., Hegedűs, P., Ladányi, G., Körtvélyesi, P., Ferenc, R., & Gyimóthy, T. (2012, September). A cost model based on software maintainability. In 2012 28th

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 06 , June - 2026

IEEE International Conference on Software Maintenance (ICSM) (pp. 316-325). IEEE.

[25] Borg, M., Udayakumar, A., & Tornhill, A. (2025, May). Industrial code quality benchmarks: toward gamification of software maintainability. In 2025 IEEE/ACM Workshop on Gamification in Software Development, Verification, and Validation (Gamify) (pp. 1-8). IEEE.

[26] Syed-Mohamad, S. M., Ngah, A., Ali, A. F. M., & Keikhosrokiani, P. (2025). Measuring Software Maintainability: An Exploration of Metrics and Continuous Practices.

[27] Tian, F., Wang, T., Liang, P., Wang, C., Khan, A. A., & Babar, M. A. (2021). The impact of traceability on software maintenance and evolution: A mapping study. Journal of Software: Evolution and Process, 33(10), e2374.