IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 1, November 2026
Impact Factor: 8.2

The Role of Generative Artificial Intelligence in Software Development

Nandini Tukaeam Ramgiri, Prof. Pawar, Prof Jadhav M.R
Vidya Niketan Institute of Management and Computer Applications, Bota, Sangamner, Maharashtra, India

Abstract : *Generative Artificial Intelligence (GenAI) is changing software development by assisting developers with code generation, debugging, documentation, test creation, and technical problem-solving. This review examines the opportunities offered by GenAI across the software development life cycle, including improved productivity, faster prototyping, and support for learning. It also discusses key challenges such as incorrect or insecure code, privacy risks, intellectual-property concerns, bias, and over-reliance on automated tools. Finally, the paper considers future perspectives, including human–AI collaboration, responsible adoption, evaluation practices, and the need for developer skills that complement AI systems.*

Keywords: *Generative Artificial Intelligence, Software Development, Large Language Models, Code Generation, Software Testing, AI Ethics*

I. INTRODUCTION

Software development involves requirements analysis, design, coding, testing, deployment, and maintenance. Generative Artificial Intelligence (GenAI), particularly tools powered by large language models (LLMs), can produce code and natural-language explanations from prompts and surrounding context. These tools are increasingly used as assistants rather than complete replacements for software engineers.

This review discusses how GenAI can support software development, the benefits and risks associated with its use, and future directions for effective and responsible adoption.

II. OPPORTUNITIES OF GENERATIVE AI IN SOFTWARE DEVELOPMENT

2.1 Code Generation and Completion

GenAI tools can suggest code snippets, complete functions, explain unfamiliar code, and help developers translate requirements into an initial implementation. This may reduce repetitive work and help teams create prototypes more quickly. Generated code should still be reviewed, tested, and checked against project requirements.

• Faster prototyping: Developers can explore ideas and produce an early working version with fewer manual steps.

• Support for learning: Explanations and examples can help learners understand programming concepts and unfamiliar APIs.

2.2 Debugging and Code Understanding

GenAI can help interpret error messages, suggest possible causes of defects, summarize existing code, and propose fixes. It can also assist when developers work with legacy systems or unfamiliar codebases. Suggestions are not always correct, so developers must reproduce the problem and validate each proposed change.

• Error analysis: AI may suggest likely causes and debugging steps based on code and error messages.

• Code explanation: Natural-language summaries can make complex functions and unfamiliar projects easier to understand.

• Refactoring support: AI can propose clearer structures or alternative implementations while preserving intended behavior.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 1, November 2026
Impact Factor: 8.2

2.3 Testing and Documentation

GenAI can assist with creating unit-test ideas, test data, comments, API documentation, and release notes. It may help identify missing test cases, but generated tests can repeat the same assumptions as the code and may overlook important edge cases.

• Test generation: Drafts of unit tests and boundary cases can help developers expand test coverage.
• Documentation: AI can prepare first drafts of technical explanations, comments, and usage examples.
• Maintenance support: Summaries of changes and code behavior can make handovers and maintenance work more efficient.

III. CHALLENGES AND LIMITATIONS

3.1 Accuracy and Reliability

GenAI systems can produce code that looks plausible but contains errors, uses outdated APIs, or fails to meet requirements. Every important suggestion should be validated through code review, automated tests, and execution in the intended environment.

3.2 Security and Privacy

Generated code may contain insecure patterns or omit important safeguards. Sharing proprietary source code, credentials, personal data, or confidential project details with an AI service may also create privacy and compliance risks. Organizations should use approved tools and follow secure development practices.

3.3 Intellectual Property and Licensing

Training data and generated output can raise questions about ownership, attribution, and software licensing. Teams should understand the terms of the tools they use and review dependencies and code provenance where required.

3.4 Bias and Unequal Performance

AI suggestions may reflect limitations or biases in training data and may work better for popular languages and frameworks than for less represented technologies. Developers should evaluate outputs in the context of their users, domain, and technical environment.

3.5 Over-Reliance and Skill Development

Relying on generated answers without understanding them can weaken debugging, design, and problem-solving practices. Developers remain responsible for architectural choices, code quality, maintainability, and the consequences of deployed software.

IV. FUTURE PERSPECTIVES

4.1 Human–AI Collaboration

Future workflows are likely to combine AI assistance with human judgment. Developers will increasingly focus on specifying requirements, reviewing outputs, making design decisions, and verifying behavior rather than accepting generated code without inspection.

4.2 Context-Aware Development Assistants

Tools may become better at understanding complete repositories, project conventions, issue histories, and build environments. Better context could improve suggestions, although access controls and data protection will remain essential.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 1, November 2026
Impact Factor: 8.2

4.3 AI-Assisted Testing and Quality Assurance

Future systems may help prioritize tests, generate diverse test cases, identify potential defects, and explain failures. Their effectiveness should be measured using reproducible evaluations and real software-quality outcomes.

4.4 Responsible and Secure AI Adoption

Organizations will need clear policies for approved tools, code review, privacy, licensing, security testing, and disclosure. Risk-based governance can help teams gain benefits while limiting avoidable harm.

4.5 Changing Developer Skills

As AI takes on more routine tasks, important skills will include problem formulation, system design, critical evaluation, secure coding, testing, and effective communication with AI tools. Continuous learning will be necessary as these systems evolve.

V. CONCLUSION

Generative AI offers practical support throughout software development, from code completion and debugging to testing and documentation. These capabilities can improve speed and accessibility, but they do not guarantee correctness, security, or maintainability. Privacy, licensing, bias, and over-reliance must be addressed through human oversight and sound engineering practices. Future progress will depend on combining capable AI tools with rigorous evaluation, responsible governance, and the expertise of software developers.

REFERENCES

1. Chen, M., et al. (2021). Evaluating Large Language Models Trained on Code. arXiv:2107.03374.
2. Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., & Karri, R. (2022). Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. IEEE Symposium on Security and Privacy.
3. Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. CHI Conference Extended Abstracts.
4. National Institute of Standards and Technology. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0). https://doi.org/10.6028/NIST.AI.100-1
5. GitHub. (2023). Research: Quantifying GitHub Copilot's impact on developer productivity. https://github.blog/