Universal Library of Engineering Technology Review Article

ISSN: 3064-996X | Volume 2, Issue 3

Open Access | PP: 29-31

DOI: https://doi.org/10.70315/uloap.ulete.2025.0203006

Prospects for the Development of Artificial Intelligence in the Field of Programming: Assessment and Forecasts According to Bolaji Olajide

Bolaji Olajide

Software Engineer, Lagos, Nigeria.

Abstract

This article analyzes the current state and prospects of applying artificial intelligence technologies in software development. It covers the main areas of AI integration into the software lifecycle: from code generation and auto-completion to automated testing, debugging, and refactoring. The purpose of the study is to evaluate the potential impact of artificial intelligence on programming processes, the role of software engineers, and the required competencies based on an analysis of existing tools, trends, and scientific publications. The objectives include systematizing data on the practical use of AI assistants, identifying advantages and limitations of their use, as well as forming forecasts about future development trajectories. The results of the study may be useful for software engineers, technical team leaders, and educational institutions to adapt to the evolving technological landscape.

Keywords: *Artificial Intelligence, Programming, Software Development, Automation, Code Generation, Machine Learning, AI Assistants.*

The rapid development of artificial intelligence technologies has a significant impact on various industries, and software development is no exception. The emergence of powerful language models and specialized AI tools opens new opportunities for automating routine tasks, increasing productivity, and changing traditional programming approaches. The relevance of this topic is driven by the need to understand the current potential and limitations of AI in this field, as well as to forecast its influence on the future of the software engineer profession. Questions arise regarding the degree of possible automation of creative and analytical aspects of development, the transformation of necessary skills, and potential changes in the labor market. The goal of this article is to analyze current AI applications in programming and assess the prospects for its further development based on scientific research and practical experience with modern AI tools.

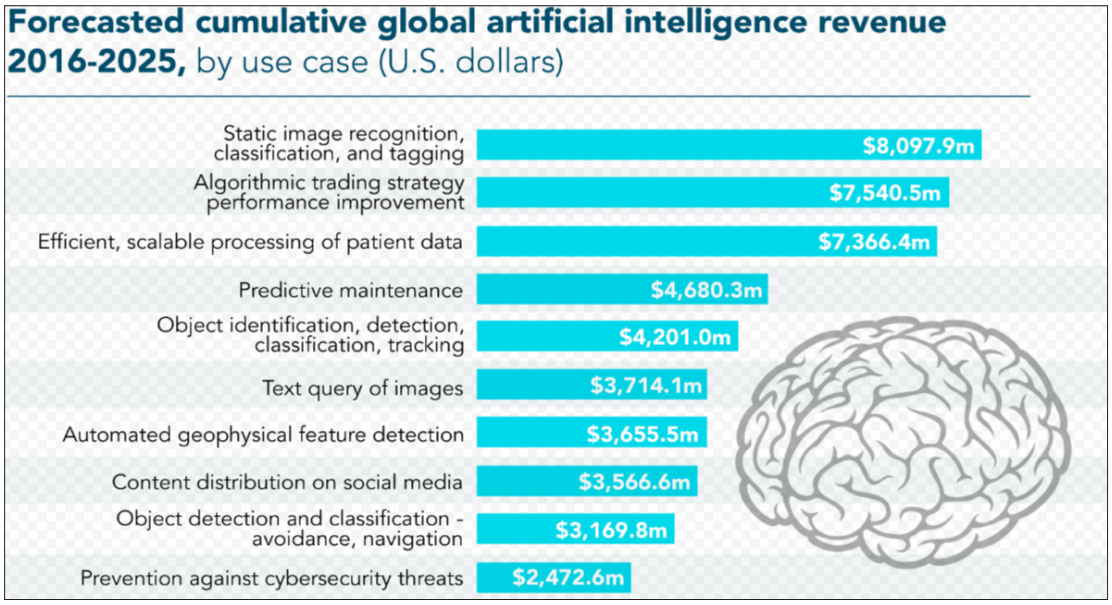
The use of artificial intelligence methods in programming has a certain history, starting from expert systems for error diagnosis to modern deep learning-based approaches. The foundation for the current breakthrough lies in advances in natural language processing (NLP) and the development of architectures like transformers, which allow models to efficiently process and generate sequences, including source code [1]. Large language models (LLMs), trained on vast corpora of text and source code, demonstrate the ability to understand context, suggest relevant code snippets, translate code between programming languages, and even explain its logic. These technologies underpin the new generation of AI programming assistants. It is important to distinguish that this concerns «weak» AI tools designed to solve specific tasks in the software development cycle, not «strong» AI capable of independent human-level creative thinking.

Modern AI technologies find applications at various stages of software creation:

- **Code generation and auto-completion:** Tools like GitHub Copilot, Amazon CodeWhisperer, Tabnine, and others analyze context (existing code, comments) and suggest line completions or entire code blocks to developers. This can significantly speed up writing template or routine code [2].
- **Automated testing and debugging:** AI is used to generate test cases, detect potential bugs and vulnerabilities at early stages. Systems can analyze bug reports, correlate them with code sections, and propose possible fixes, reducing time spent on defect identification and resolution [3].

Citation: Bolaji Olajide, "Prospects for the Development of Artificial Intelligence in the Field of Programming: Assessment and Forecasts According to Bolaji Olajide", Universal Library of Engineering Technology, 2025; 2(3): 29-31. DOI: https://doi.org/10.70315/uloap.ulete.2025.0203006.

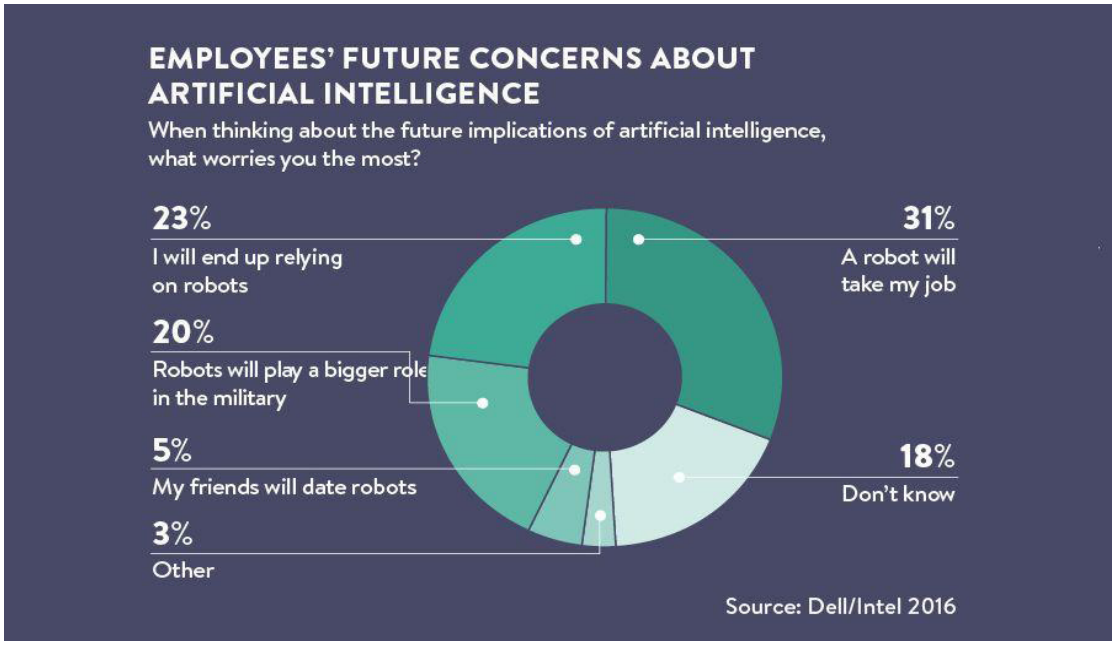
- **Code refactoring and optimization:** AI algorithms can analyze existing codebases, identify code smells, and suggest refactoring options to improve readability, maintainability, and performance.
- **Requirements analysis and documentation:** NLP methods are applied to analyze technical documentation, extract requirements, and even generate basic documentation based on source code.



Integration of AI into the development process leads to a transformation of the software engineer's role. The share of time spent writing standard code decreases, while the importance of higher-level tasks grows: designing system architecture, setting tasks, critically evaluating and verifying AI-generated code, as well as solving non-trivial problems. Developers need to develop new competencies:

- **Skills in interacting with AI (Prompt Engineering):** The ability to formulate precise and effective queries to AI models to obtain the desired results.
- **Critical thinking and verification:** The ability to analyze and verify the correctness, security, and efficiency of code suggested by AI, since models can generate errors or suboptimal solutions [4].
- **Understanding AI principles:** Basic knowledge of the limitations and capabilities of AI tools to apply them appropriately.
- **Systems thinking and architectural skills:** The focus shifts from writing individual code snippets to designing holistic and reliable systems.

There are concerns about a possible reduction in demand for junior specialists, whose work mainly involves routine tasks easily automated by AI. However, as research shows [5], in the near future AI is more likely to be seen as a productivity-enhancing tool for existing specialists rather than a full replacement.



Further development of AI in programming will likely follow the path of deeper integration and expanded functionality of tools. We can expect the emergence of more autonomous systems capable of handling complex tasks such as creating entire modules based on high-level descriptions or proactively detecting and fixing errors in real time. Improvements in AI's ability to understand complex project contexts and long-term dependencies in code are also forecasted.

However, fundamental limitations remain. Current AI models struggle with true semantic understanding, creativity in solving fundamentally new problems, and adapting to rapidly changing requirements without retraining [6]. Full automation of the entire software development process—especially in task definition, complex system design, and consideration of implicit client requirements—seems unlikely in the medium term. The most realistic scenario is a human-AI symbiosis, where AI acts as a powerful assistant that frees engineers from routine work and allows them to focus on strategic and creative aspects.

Artificial intelligence is becoming an integral part of the modern software engineer's toolkit. AI technologies demonstrate significant potential for automating code generation, testing, debugging, and refactoring tasks, contributing to increased productivity and reduced development time. At the same time, using AI requires specialists to develop new skills related to interacting with models, critically evaluating their outputs, and shifting focus toward design and architecture tasks.

The current state of technology and expert opinion analysis [see 4, 5] suggest that in the foreseeable future, AI will serve to augment human capabilities rather than replace them completely. The software engineer profession is transforming, but not disappearing. The ability to adapt, master new tools, and focus on tasks requiring deep understanding, creativity, and systems thinking will determine specialists' success in the new technological reality. Practical recommendations include active learning and integration of AI tools into workflows with mandatory quality and security controls over generated code, as well as revising educational programs to prepare engineers for new market demands.

REFERENCES

1. Vaswani A. et al. Attention is all you need // Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008.
2. Dakhel A. M. et al. GitHub Copilot AI pair programmer: Asset or Liability? // 2022 IEEE/ACM 45th International Conference on Software Engineering: Software
3. Engineering in Practice (ICSE-SEIP). Melbourne, Australia: IEEE, 2022. P. 316–326. Just R., Schweiggert F., Fraser G. Generating Unit Tests with Debugger-Based Program Analysis // 2021 IEEE International Conference on Software Testing, Verification and Validation (ICST). Online: IEEE, 2021. P. 13–24.
4. Pearce H. et al. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions // 2022 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA: IEEE, 2022. P. 754–768.
5. Ford D. et al. A Case for AI-Augmented Software Engineering // Queue. 2023. Vol. 21, № 1. P. 40–65.
6. Marcus G. The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence // arXiv preprint arXiv:2002.06177. 2020. URL: https://arxiv.org/abs/2002.06177