241. THE IMPACT OF CONTEXT-AWARE ARTIFICIAL INTELLIGENCE ON MODERN SOFTWARE DEVELOPMENT LIFECYCLES

Egorov A.S.
Belarusian State University of Informatics and Radioelectronics
Minsk, Republic of Belarus

Shaputsko A.V. – Lecturer, Master of Arts (Philology)

This paper examines the impact of context-aware AI coding tools, analyzing both their advantages and disadvantages to provide a fair perspective on AI in software engineering.

The integration of artificial intelligence in software development has extremely changed in recent years, from simple autocompletion of code and hints to advanced context-sensitive development tools. In the modern software development world of 2024-2026, code assistances such as Cursor, GitHub Copilot, Claude, and new one such as Antigravity have made a significant contribution to the process of analyzing, improving and maintaining source code. The first AI models could only view a few lines of code to predict the next operators. Modern artificial intelligence tools use large "context" and advanced search methods to read and understand the entire architecture of a project.

This evolution from simple local suggestions to system work has completely changed the daily work of developers. Instead of manually typing each individual line of simple code, the engineer can focus on high-level problems in the relevant project. Since artificial intelligence models are already embedded in the Integrated Development Environment (IDE), they can analyze how different parts of the software interact with each other. They understand how the user interface connects to the database, how data models interact, and what security rules must be implemented. This understanding allows AI to act not as a simple dictionary assistant, but rather as a highly qualified programming partner.

When a developer prompts artificial intelligence to create a new feature, the machine learning model does not produce an isolated, random piece of code. Instead, it provides a complete set of changes covering many connected files and directories. This deep contextual integration significantly reduces the time required to write monotonous, repetitive code or rewriting standard algorithms. Moreover, artificial intelligence can instantly give information on specific programming syntax and platform details, without requiring developers to search the entire Internet or read external documentation to solve problems.

Research analyzing developer productivity demonstrates a significant reduction in the timeframe from an idea to a working prototype [1]. Teams that integrate context-sensitive assistants point out that they can release ready-to-deploy software much faster. By delegating simple programming issues to artificial intelligence, engineers get more free time to solve more complicated tasks, such as developing a better software architecture, solving complex logic problems, and improving user interaction. In general, it increases the overall quality of software being created. However, despite these advantages, the extensive capabilities of context-sensitive artificial intelligence tools are associated with a number of serious operational, logical and financial risks that can negate all the promised performance gains.

One of the most dangerous side effects of using highly autonomous AI coding agents is the loss of software quality control and security. This phenomenon is associated with a bias towards automation, where developers tend to trust automated systems. When engineers see that AI generates code quickly and without syntaxes errors, they begin to blindly trust the machine. Instead of carefully reading and testing the generated code, specialists often simply look it through quickly. If the code works under the basic “happy path” scenario, they approve it, completely ignoring more complex cases or hidden security flaws. This behavior can lead to significant security vulnerabilities. The study, conducted by researchers at Stanford University, evaluated developers who use artificial intelligence programming assistants, such as Open AI's Codex, in security-related programming tasks. The results showed that participants who had access to the help of artificial intelligence created significantly less secure code than those in the control group who coded manually [1]. For example, when implementing an encryption function, developers often approve insecure solutions offered by AI. Additionally, the study showed that developers using artificial intelligence reported higher confidence in the security of their code, despite its objectively lower security quality [1]. The professional look at AI-generated code tricks developers with a false sense of security, forcing them to implement software with vulnerabilities in production environments where hackers can crack it.

In addition to generating insecure logic, context-aware AI tools also have the ability to invent things that do not actually exist. Considering that large language models create text by predicting the most likely next token based on patterns in their training data, they can confidently generate dummy URLs, documentation, or even software libraries. In the software development community, this phenomenon is often referred to as hallucination of artificial intelligence, and it poses a significant risk to software development. In 2023, the

cybersecurity research team at Vulcan Cyber demonstrated how serious this problem can be. They evaluated several generative AI tools and found that in almost 30 % of coding-related queries, the machine learning model recommended at least one software package that did not exist in any legitimate repository [2]. Attackers can take advantage of this behavior by requesting coding solutions from artificial intelligence systems, identifying fabricated package names, and then publishing malicious packages under the same names. To demonstrate the possibility of this attack, Vulcan Cyber researchers uploaded an empty, harmless package, using a name that the AI often came up with hallucinations. Within three months, the package received more than 30,000 legitimate downloads from real developers. This case illustrates how developers can rely on suggestions generated by AI without verifying whether the mentioned libraries or dependencies actually provides by verified sources. As a result, attackers can potentially gain access to the developer environment and, in some cases, to the corporate software infrastructure.

Another common problem for developers occurs when system tools based on neural networks make unnecessary changes to the project codebase. When a programmer asks a context-oriented utils, such as Cursor or Claude, to implement a small function, the system analyzes the surrounding files and, in an attempt to optimize the solution, can start rewriting code that is already functioning. This behavior, sometimes described as “unexpected refactoring” can lead to replacing existing logic with alternative implementations or restructuring project features that did not require modification. As a result, developers often end up with large sets of code changes that contain numerous undocumented edits. In the future, this makes it difficult to determine which parts of the code implement the intended function, and which have been modified by artificial intelligence. These changes can also disrupt the historical structure of the project code, making it difficult for future developers to understand previously made decisions.

In addition, allowing AI tools to automatically modify multiple files carries the risk of destructive changes: a poorly phrased prompt can cause the system to misinterpret the intended task and change or delete important logical or, even worse, configuration files. Since such tools work very quickly, even a small human error can lead to large-scale changes in the repository, forcing later development teams to spend a great deal restoring previous versions.

Finally, developers and companies face economic and performance challenges when using context-sensitive artificial intelligence tools. To analyze large software projects, modern systems rely on extensive context that can process hundreds of thousands or even millions of tokens (segments of text or code) at once. While this feature allows the model to view large sections of project code, it also requires significant computational resources and increases processing time. These deeply analyzing context operations are expensive, as current models charge fees based on the number of tokens processed. Recent cost analyses of enterprise artificial intelligence suggest that scanning a large repository to complete a single complex query can cost several dollars. When developers start constantly using such features to perform routine tasks, these costs quickly accumulate. As a result, companies may discover that the financial and computational costs of large-scale contextual analysis reduce or even exceed the performance benefits of faster code generation.

In summary, context-aware artificial intelligence tools such as Cursor, GitHub Copilot, Claude, and Antigravity provide additional opportunities for software development. Their ability to understand the architecture of a large-scale project, generate changes across multiple files, and reduce repetitive coding tasks allows developers to focus on high-level design, complex logic, and improving user experience. These benefits will result in faster development cycles, higher productivity, and potentially higher software quality. At the same time, the benefits come with significant disadvantages. Excessive use of artificial intelligence can cause developers to overlook important security issues. Hallucinations with artificial intelligence packages pose a real threat to software supply chains, while unexpected refactoring and automatic changes to multiple files can compromise the maintainability of code. In general, the computational and financial costs of analyzing a large context can limit the economic benefits of software development scalability. Although these risks are significant, they can and should be controlled. Teams that maintain careful review practices of code generated by artificial intelligence, and remain vigilant about potential bugs can minimize security vulnerabilities and coding errors. Therefore, context-oriented artificial intelligence can be used effectively, speeding up development while maintaining the reliability and security of software.

References:

1. Do users write more insecure code using AI assistants? – arXiv. – URL: https://arxiv.org/abs/2211.03622://arxiv.org/abs/2211.03622 (date of access: 05.03.2026).

2. Hallucination of the artificial intelligence package: a new attack technique – VulcanCyber. – URL: https://vulcan.io/blog/ai-hallucinations-package-risk://vulcan.io/blog/ai-hallucinations-package-risk/ (date of access: 10.03.2026).

3. How I use Claude Code to accelerate my software engineering job and improve my life. – URL: https://dev.to/juandj/how-i-use-claude-code-to-accelerate-my-software-engineering-job-and-improve-my-life-8o7 (date of access: 12.03.2026).