***O. Voloshyn**, BA student*
***O. Dienichieva**, PhD in Ped., As. Prof.*
Zhytomyr Polytechnic State University

AI DEPENDENCY TRAP IN MODERN SOFTWARE DEVELOPMENT

Software engineering is changing fast right now, mostly because AI-powered coding tools have become so incredibly common. By mid-2025, a massive 84% of professional developers said they were either already using or planning to use AI in their daily work [1]. To put that in perspective, GitHub Copilot alone is writing nearly half (46%) of the code for its 20 million users, and 90% of Fortune 100 companies have adopted it [2]. But this huge rush to use AI hides some real problems. Even though these tools help people work faster, 46% of developers actually do not trust the code they get back. Their biggest complaint is that the AI produces code that is almost right, but not quite correct[1]. This contradiction points to some hidden dangers in relying too heavily on automated code generation**.**

The biggest issue coming out of all this is what we can call the "AI Dependency Trap." It happens when developers stop using AI just as a helpful assistant and start using it as a crutch to do the hard thinking for them. In the past, struggling through tough logic problems and cryptic errors was exactly how developers built their deep problem-solving skills [1]. But when you offload that thinking to an AI, you lose that "productive struggle" [7]. Developers can write code quickly, but they often do not really grasp the underlying architecture [3]. This growing reliance on AI is causing a drop in independent analytical skills. It's also quietly building up technical debt because, while AI can write a quick function, it usually lacks the judgment needed to design a stable, system-wide architecture [3].

A major psychological side effect of leaning too hard on AI is the illusion of competence. Basically, this is a cognitive bias where developers trick themselves into thinking they understand the code just because they know how to write a good prompt [7]. Research shows that while AI helps developers finish tasks much faster, their actual understanding of the project drops significantly if they just let the AI handle all the logic [7]. Because the AI generates code so smoothly, it gives a false sense of mastery. Developers don't put in the mental effort required to actually remember or understand what was built, and they end up pushing code into production that nobody fully understands [3].

This false confidence naturally leads to nightmare debugging scenarios. When AI-written code inevitably breaks, fixing it takes way longer because the developer never really thought through the logic in the first place. In fact, a 2025 study found that for complex tasks, using advanced AI tools actually slowed developers down by 19% [5]. This happened because they had to pay a massive "debugging tax" - spending hours verifying and fixing code that looked right but was not.

On top of that, AI struggles to understand complex environments. When researchers tested 300 AI-generated projects, they found huge gaps in whether the code could actually run. While Python code worked about 89.2% of the time, Java code only worked 44.0% of the time [6]. A big part of the problem is that AI often underestimates what a project needs to run. It might claim you only need three dependencies, but in reality, the system requires 37, creating a tangled mess of hidden complexity [6].

There is no denying that artificial intelligence is an incredible tool for automating repetitive tasks and writing boilerplate code faster. However, the data make it clear that AI amplifies existing team practices; it is not an independent architect capable of fixing a broken system [4]. The AI Dependency Trap is a real threat to the long-term health of software because it replaces deep, critical thinking with simple prompting. If we want to use generative AI safely, developers absolutely need to keep their core coding and architectural skills sharp. Building reliable software in the future won't be about outsourcing our thinking to machines, but rather using AI to support a strong, human-driven architectural vision.

REFERENCES

1. Stack Overflow, 2025 Developer Survey, Stack Overflow, 2025. [Online]. Available: https://survey.stackoverflow.co/2025/ai
2. QuantumRun, GitHub Copilot Statistics and Market Adoption, QuantumRun Consulting, 2025. [Online]. Available: https://www.quantumrun.com/consulting/github-copilot-statistics/
3. E. Paz, Army of Juniors: The AI Code Security Crisis, OX Research, 2025. [Online]. Available: https://www.ox.security/wp-content/uploads/2025/10/Army-of-Juniors-The-AI-Code-Security-Crisis.pdf
4. N. Harvey, and D. DeBellis, State of AI-assisted Software Development, Google Cloud, 2025. [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/announcing-the-2025-dora-report
5. J. Becker, N. Rush, B. Barnes, and D. Rein, Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity, Model Evaluation & Threat Research (METR), 2025. [Online]. Available: https://arxiv.org/pdf/2507.09089
6. B.P. Vangala, A. Adibifar, A. Gehani, and T. Malik, AI-Generated Code Is Not Reproducible (Yet), arXiv, 2025. [Online]. Available: https://arxiv.org/pdf/2512.22387
7. F. Bolici, K. Crowston, A. Varone, and M. Fudge, Rethinking Programming Skills in the Age of Generative AI, Proceedings of the 59th Hawaii International Conference on System Sciences, 2026. [Online]. Available: https://hdl.handle.net/10125/112269