The Near Future of AI: From Code Generation to Context Engineering

Hector Lugo III

Position

We're watching AI evolve from a fancy autocomplete into something closer to a thinking partner. The next big leap won't come from throwing more computing power at bigger models—it'll come from teaching AI how to actually *work* with us. I believe this shift toward what I call *context engineering* is going to fundamentally change not just how we write software, but how we think about collaboration between humans and machines.

Background

Tools like GPT-4, Claude, and Gemini have already made AI coding assistants a daily reality for millions of developers. What makes code such a natural fit for AI? Simple: you get instant feedback. Run the code, see if it breaks, adjust accordingly. This tight feedback loop is perfect for what researchers call "agentic AI"—systems that can plan their next move, try things out, and learn from mistakes [1, 2].

But here's where it gets interesting: even with all this progress, AI still trips over the messy realities of actual software projects. MIT's Computer Science and Artificial Intelligence Laboratory ran a study showing that current models struggle badly with large-scale refactoring, complex debugging, and working within the quirks of real-world codebases [3]. The problem isn't raw intelligence—it's that these systems lack the surrounding context and structure they need to truly collaborate with human teams.

Argument

Context engineering flips the script on how we work with AI. Instead of typing one-off prompts and hoping for the best, we're building entire *environments* where AI can operate: documentation that explains what we're trying to accomplish, structured plans that break down complex work, validation systems that catch mistakes before they matter. It's like the difference between telling someone "fix the bug" versus giving them the codebase, test suite, project roadmap, and a cup of coffee.

This approach makes AI dramatically more reliable and scalable. Rather than guessing what you meant, an AI can reference actual project artifacts. Teams can deploy multiple specialized AI agents that stay coordinated through shared context. We're already seeing this in action with tools like GitHub Copilot's Model Context Protocol and Anthropic's Claude Code—systems that can read your files, run commands, and build on their own earlier work.

But this shift raises uncomfortable questions about what "software engineering" even means going forward. Google DeepMind's CodeMender agent can now automatically hunt down and patch security vulnerabilities [4]. Google's Gemini AI recently solved a competition problem

that stumped over a hundred human teams at the International Collegiate Programming Championship [6]. Meta's experimenting with gamified dashboards to track how much developers use AI tools [5].

What happens when AI handles the grunt work? My bet is that developers evolve into something closer to architects and strategists—people who frame problems, design systems, and create the conditions where AI can do its best work. The value shifts from raw coding ability to judgment, creativity, and the ability to structure complex challenges.

Open Questions

1. How do we even measure whether an AI has "good enough" context? What makes an environment clear, complete, and trustworthy for an autonomous agent?
2. If AI can handle most coding tasks, will the industry start valuing adaptability and problem-framing skills over years of language-specific experience?
3. Who's responsible when an autonomous agent introduces a critical bug—or worse, a security flaw—into production code? What ethical guardrails do we need before this becomes routine?

Further Reading

- MIT CSAIL (2024). *Challenges and Paths Towards AI for Software Engineering.* MIT News
- Anthropic Engineering Blog. Claude Code Best Practices
- GitHub Copilot Docs. Copilot Best Practices
- Google DeepMind (2025). *CodeMender: AI Agent for Automated Vulnerability Fixes*
- Google (2025). *Gemini AI Solves Coding Problem that Stumped 139 Human Teams at ICPC World Finals*

Conclusion

The future I see isn't about AI replacing programmers—it's about reimagining the workspace where humans and AI collaborate. As context engineering becomes as fundamental as version control or testing, the craft of software development will shift from writing lines of code to orchestrating intelligent systems. Code stops being just instructions for computers and becomes a shared language—a bridge between human intuition and machine precision. That's the future worth building toward.

References

[1] Anthropic. (2024). Claude Code Best Practices. Retrieved from https://www.anthropic.com/engineering/claude-code-best-practices

[2] GitHub. (2024). Copilot Best Practices. Retrieved from https://docs.github.com/en/copilot/get-started/best-practices

[3] MIT CSAIL. (2024). Challenges and Paths Towards AI for Software Engineering. Retrieved from https://news.mit.edu/2024/ai-software-engineering-roadblocks-0927

[4] Google DeepMind. (2025). CodeMender: AI Agent for Automated Vulnerability Fixes. Retrieved from https://www.deepmind.com/blog

[5] Meta AI. (2025). Monitoring AI Use Through Workplace Gamification. Retrieved from https://www.businessinsider.com/meta-ai-gamified-adoption-2025

[6] Google. (2025). Gemini AI Solves Coding Problem that Stumped 139 Human Teams at ICPC World Finals. Retrieved from https://www.theverge.com/2025/03/15/gemini-ai-icpc-results