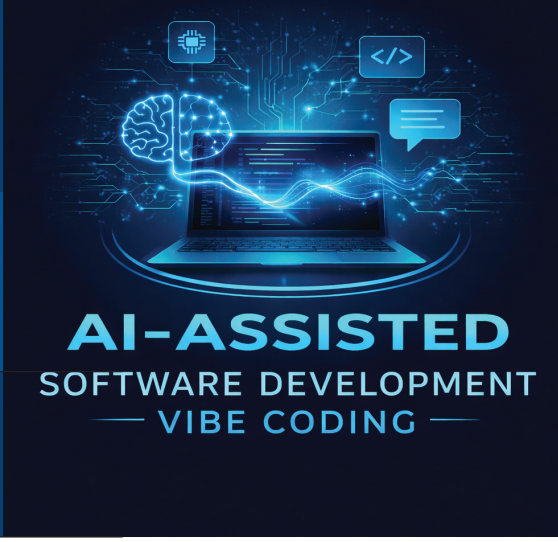
Association for Computing Machinery

acm Technology Policy Council

TechBriefs

AI-Assisted Software Development, or Vibe Coding: Benefits and Risks of AI-Driven Software Development

by Simson Garfinkel, Mohan Sankaran, Rohan Sharma, Shrinivass Arunachalam Balasubramanian, Arpan Pandey, and Aruun Kumar



AI-Assisted Software Development, often referred to as "Vibe Coding," is the practice of using Generative Artificial Intelligence to create or modify software systems in which humans describe what they want to build or modify, and an AI coding assistant writes and debugs computer code. Several popular vibe coding systems are built on top of Agentic AI systems, an "approach of making AI systems capable of setting or refining plans and executing tasks with minimal or no human oversight" [1].

Vibe Coding Benefits

Vibe coding enables people with little or no coding experience to create highly functional applications [2]. It can also assist experienced programmers by generating code that leverages complex application programming interfaces (APIs), a hallmark of modern software development.

Because vibe coding lets developers spend less time writing code, they can focus on higher-level concerns like design, user experience, and other creative problem-solving. Vibe coding might thus shift developer effort from time-consuming implementation toward higher-level design and intent specification.

Many developers report feeling more productive when using AI to generate code [3], especially with mundane programming tasks that do not require significant creativity [4], although these reports are subjective and may not be borne out by empirical measurements over time.

Vibe Coding Risks

Software engineering's established practices produce systems that are generally secure, reliable, and maintainable. Vibe coding circumvents these practices. While it can produce code that meets immediate requirements for style, conventions, and targeted ("unit") tests, it does not produce well-designed software systems. Because many of these systems have been trained on data that includes cybersecurity vulnerabilities, there is a risk that they will replicate these in the code that they generate [5, 6].

A core principle of modern software development is that a program's functions and behavior need to be specified in advance. "A program that has not been specified cannot be incorrect, it can only be surprising" [7]. AI-generated code typically lacks specifications. Even when specifications are provided, many of today's vibe coding platforms lack mechanisms to enforce them. As a result, AI-generated code drifts away from stated requirements, including core functionality.

Few vibe coding platforms systematically test their AI-generated code to ensure it runs correctly and consistently [8]. Although it is possible to give these systems acceptance tests for the code they generate—or even have them generate their own tests—AI systems have been observed to modify, disable, or simply remove such tests rather than correcting their code [9, 10].

Vibe coding platforms often produce over-engineered solutions with redundant code and subtle errors that create maintenance nightmares, known as "technical debt" [11]. Entry-level programmers do this as well, but they are typically supervised by senior programmers when code is critical. Entry-level programmers often seek to improve their skills and are penalized if they try to subvert internal controls. AI-generated code, in contrast, is frequently unaudited, and there

is no way to penalize a misbehaving AI. This can result in code that is, paradoxically, maintainable only by AI: the sheer volume and complexity of AI-generated code make manual code review impractical, increasing the likelihood that undetected errors slip into production.

Recently, many vibe coding platforms have added "agentic" features that go beyond software development, allowing the platform to run programs on the software developer's behalf, often without the human first reviewing and approving the program's execution. This can make users more productive, since the platform can operate more quickly without human intervention. However, it also lulls the user into granting the platform increased authority to run new executables without explicit review.

The agentic platforms can typically execute these programs not only on users' computers but also on any computer reachable over their network. This leaves the users and their networks at risk if the AI executes commands users did not intend. For example, deleting critical information, sending confidential information outside the enterprise security perimeter, downloading and executing software from the Internet, or reconfiguring computers so they become susceptible to intrusion. Vibe coding platforms can also be vulnerable to "prompt injection attacks" when third parties embed malicious commands in software that are interpreted as instructions from the programmer [12].

Vibe coders may generate significantly more CO_2 emissions than traditional programmers. This is often debated, as vibe coding produces code faster than humans do, and in small-language models, the total energy difference between AI and prolonged code development could be comparable. But because vibe coding often overproduces code, it still requires human intervention to refine and optimize. Energy consumption with "standard, widely-used models is far more environmentally strenuous" [13].

Vibe coding may also have long-term negative effects on skill development in the programming profession. An internal study from a major AI provider found that students and early-career programmers using vibe coding showed decreased mastery of sophisticated programming concepts and skills [14]. In educational settings, students with advanced programming skills were more likely to succeed in building a program with AI assistance, whereas students with less coding experience were less likely to do so, indicating that instruction in fundamental programming concepts remains necessary.

Vibe coding may thus contribute to a hypothesized "experience gap," in which AI automates many early-career skills that are both drudgery for more experienced programmers and a necessary step in building mastery. Such skills include simplifying redundant code, porting code to new environments, and the routine addition of simple features, which typically require a programmer to first understand the codebase. Some studies have shown significant cognitive erosion resulting from AI tools, although they did not specifically consider vibe coding [15, 16]. Nevertheless, by eliminating opportunities for junior programmers to become senior while simultaneously deskilling those later in their careers, increased AI use in software development may paradoxically contribute to a shortage of more experienced workers.

Conclusion

It is unclear what vibe coding means for the future of programming or the economic outlook for the programming profession. While the job market for programmers appears to be cooling [17], some studies find that junior developers see the biggest impact of vibe coding, which makes it less likely they will themselves be replaced with AI agents [18].

Vibe coding can make expert developers more productive and allow novice developers to create and deploy working apps, but current platforms do not enforce modern software engineering practices. The core issues are systemic: these platforms do not create formal specifications and frequently ignore them when provided; they do not systematically test their outputs and may remove/modify failing tests rather than address the underlying problems; and they generate code that becomes maintainable only by AI, not by human developers. The same mechanism responsible for these failures — the lack of a rigorously enforced semantic model that allows AI systems to validate their outputs — is also responsible for AI hallucinations more broadly. Because of these fundamental limitations, vibe coding requires that users and organizations compensate with improved technical checks and governance mechanisms to avoid predictable failure modes.

Existing techniques for improving code quality can be applied to both human- and AI-generated code. This includes the use of mathematical verification and other formal methods and techniques [19], as well as new work on developing specially tuned AI models adept at finding security vulnerabilities [20]. Such techniques will be needed to make vibe coding a cost-effective and secure alternative to traditional software development.

NOTES AND SOURCES

1. Alejandro Bellogin *et al.* 2025. Systemic Risks Associated with Agentic AI: A Policy Brief. ACM Europe TPC Autonomous Systems Subcommittee. Retrieved March 27, 2026 from https://www.acm.org/binaries/content/assets/public-policy/europe-tpc/systemic_risks_agentic_ai_policy-brief_final.pdf
2. Jesse G. Meyer. 2026. Vibe Coding Omics Data Analysis Applications. *Journal of Proteome Research* 25, 2 (Jan. 2026), 1191–1197. https://doi.org/10.1021/acs.jproteome.5c00984
3. Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2024. Measuring GitHub Copilot's Impact on Productivity. Communications of the ACM 67, 6 (Jun. 2024), 68–77. https://doi.org/10.1145/3633453
4. Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to Help With Code Understanding. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal. IEEE, 1–13. https://doi.org/10.1145/3597503.3608128
5. Andrew Stiefel. 2025. Understanding Security Risks in AI-Generated Code. Cloud Security Alliance. Retrieved March 27, 2026 from https://cloudsecurityalliance.org/blog/2025/07/09/understanding-security-risks-in-ai-generated-code
6. J.-P. Joosting. 2025. Report Finds AI-Generated Code Poses Security Risks. eeNews Europe. Retrieved March 27, 2026 from https://www.eenewseurope.com/en/report-finds-ai-generated-code-poses-security-risks
7. W. D. Young, W. E. Boebert, and R. Y. Kain. 1985. Proving a Computer System Secure. The Scientific Honeyweller 6, 2 (Jul. 1985), 18–27.
8. Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming Jiang. 2023. GitHub Copilot AI Pair Programmer: Asset or Liability? Journal of Systems and Software 203 (2023), 111734. https://doi.org/10.1016/j.jss.2023.111734
9. H. Rojas-Galeano. 2025. The Vibe-Check Protocol: Quantifying Cognitive Offloading in AI Programming. arXiv. https://doi.org/10.48550/arxiv.2601.02410
10. Addy Osmani. 2025. Vibe Coding Is Not the Same as AI-Assisted Engineering. Retrieved March 27, 2026 from https://medium.com/@addyosmani/vibe-coding-is-not-the-same-as-ai-assisted-engineering-3f81088d5b98
11. Bill Doerrfeld. 2025. How AI Generated Code Compounds Technical Debt. LeadDev. Retrieved March 27, 2026 from https://leaddev.com/technical-direction/how-ai-generated-code-accelerates-technical-debt
12. John Cranney. 2025. Prompt Injection and the Security Risks of Agentic Coding Tools. Secure Code Warrior. Retrieved March 27, 2026 from https://www.securecodewarrior.com/article/prompt-injection-and-the-security-risks-of-agentic-coding-tools
13. N. H. Woo. 2025. A Comparative Study of AI and Human Programming on Environmental Sustainability. Scientific Reports 15 (2025), 39182. https://doi.org/10.1038/s41598-025-24658-5
14. Judy Hanwen Shen and Alex Tamkin. 2026. How AI Impacts Skill Formation. arXiv. https://arxiv.org/abs/2601.20245
15. B. Jose, J. Cherian, A. M. Verghis, S. M. Varghise, M. S, and S. Joseph. 2025. The Cognitive Paradox of AI in Education: Between Enhancement and Erosion. Frontiers in Psychology 16 (Apr. 2025), 1550621. https://doi.org/10.3389/fpsyg.2025.1550621
16. Michael Gerlich. 2025. AI Tools in Society: Impacts on Cognitive Offloading and the Future of Critical Thinking. Societies 15, 1 (2025), 6. https://doi.org/10.3390/soc15010006
17. Esther Shein. 2025. The Outlook for Programmers. Commun. ACM 68, 5 (May 2025), 12–14. https://doi.org/10.1145/3715692
18. Manuel Hoffmann, Sam Boysel, Frank Nagle, Sida Peng, and Kevin Xu. 2025. Generative AI and the Nature of Work. Harvard Business School Strategy Unit Working Paper No. 25-021. https://doi.org/10.2139/ssrn.5007084
19. Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. 2022. Toward Verified Artificial Intelligence. Communications of the ACM 65, 7 (2022), 46–55. https://doi.org/10.1145/3503914
20. Catherine Tony, Nicolás E. Díaz Ferreyra, Markus Mutas, Salem Dhif, and Riccardo Scandariato. 2025. Prompting Techniques for Secure Code Generation: A Systematic Investigation. ACM Trans. Softw. Eng. Methodol. 34, 8, Article 225 (November 2025), 53 pages. https://doi.org/10.1145/3722108

Note: Several references are to arXiv preprints. These papers have not undergone peer review yet, but are included here for completeness and readability, reflecting the rapid pace of research in this field.

ADDITIONAL INFORMATION

With 100,000 members in 190 countries, the nonprofit **Association for Computing Machinery** is the world's largest and longest-established organization of professionals involved in all aspects of computing. Under the auspices of the global ACM Technology Policy Council, policy committees in the U.S. and Europe provide cutting-edge, apolitical, non-lobbying information about computing and its social impacts to policy makers at all levels of government in many forms, including briefings, testimony, consultation, and rulemaking comments, reports, and analyses.

To tap the deep expertise of ACM's global membership, please contact ACM's Global Policy Office at *acmpo@acm.org* or +1 202.580.6555. To receive ACM TechBriefs quarterly, in the body of a one-line email send *subscribe ACM-tpc-tech-briefs* followed by your first and last names to *listserv@listserv.acm.org*

ACKNOWLEDGMENTS

The authors would like to thank the following subject-matter experts for their reviews and input: Dr. Dan Wallach, Rice University; Dr. Eelco Herder, Utrecht University; Dr. Michel Beaudouin-Lafon, Université Paris-Saclay; Dr. Nicholas Mattei, Tulane University. A thank you to the ACM TechBrief Committee and the ACM Technology Policy Council for their valuable input. The PDF version of the document meets WCAG 2.2 Level AA accessibility standards and complies with the PDF Accessibility Checker (PAC2024) requirements.

DOI: https://doi.org/10.1145/3807517.3807518 ISBN: 979-8-4007-3219-5