De-scripting Coding Assistants in the Open Source Community

Shepard Rodgers

STS-150: Automation and AI in Historical Context

May 2, 2025

Before it was a programmatic machine, the computer was an occupation; a person who performed mathematical calculations without the help of an electronic calculator. History is littered with examples of technologies being named after the jobs or tasks they were intended to replace: ATMs (Automatic Teller Machines), Autopilot, apps like "Messenger," and even Robots (from Czech, "forced labor").[1] However, this is not a one-way interaction. Jessa Lingel describes how the roles of the secretary and typewriter were born from their homonymous precursors, a desk and a writing machine, with the politics of these technologies being inscribed into the occupations.[2]

Today, this reciprocal approach to defining labor and machines can be traced to open-source software development. This ecosystem is built on mutual contributions and community-driven programming, with the Pull Request (PR) process providing contributors with specific feedback on their code and projects with a robust quality assurance system. Recently, "coding assistants" powered by frontier AI models have been envisioned as a fill-in for the collaboration that supports open source projects. GitHub Copilot, for example, is named after the role of a human collaborator in the pair programming process. But Copilot is not a peer—it is software, embedded into the developer's environment, offering constant assistance. Here, the intended purpose of the AI assistant is to transform programming into a seamless, integrated, and individual process, without an explicit need for human collaboration or feedback. With a coding assistant, one programmer is suddenly expected to do the work of many. According to this new regime, they should be faster, more productive, and less reliant on human mentorship. These assumptions about how the open-source community should be reshaped to match the expectations of AI-assisted coding fit Madeleine Akrich's definition of a "script," as they

[1] Oxford English Dictionary, "robot (n.1), Etymology," July 2023, https://doi.org/10.1093/OED/4915451935.
[2] Lingel, Jessa. "'Alexa, Tell Me about Your Mother': The History of the Secretary and the End of Secrecy." Catalyst, 2020. https://doi.org/10.28968/cftt.v6i1.29949.

describe beliefs about the world into which the technology will be inserted and how the people in this world will interact with it.[3]

However, this vision ignores the value of collective contribution in open source development. One of the core motivations behind this unpaid labor is the opportunity to interact with a community of developers who are equally passionate about the projects they are developing. These interactions are a chance to critique code, learn about areas for improvement, and discuss ideas for a project. When open source developers are siloed into individual development and expected to produce more code using a flawed and constantly shifting technology, they will necessarily be driven to push back. Looking at historical examples of how speedup on the factory floor and surveillance in digitally-organized work have been resisted, a new path forward appears for open source developers hoping to maintain the constructive collaboration of their work while benefiting from the ubiquity of AI coding tools like Copilot.

Copilot and the Reimagined Programmer

The early computers of the 1940s could only be programmed in their native language: 1s and 0s. The need for more intuitive forms of computer programming led to the development of assembly language in the late 1940s, translating human-readable instructions into binary. The 50s and 60s saw the rise of compiled languages such as Autocode, FORTRAN, and BASIC, where the human-written source code would be interpreted as machine code before execution. This drive to build more user-friendly and powerful programming languages continued for decades, giving rise to the “high-level” languages of today such as Python and JavaScript, which are built

[3] Madeleine, Akrich. “The De-Scription of Technical Objects.” Shaping Technology/Building Society: Studies in Sociotechnical Change, MIT Press, 1992.

on countless incremental improvements and layers of abstraction to bring computer programming closer to natural human language.[4] For technologists, the next logical step in this progression is the ability to write code using one's native language, without the need to learn the details of coding syntax or style. This is how we arrive at the well-intentioned integration of Artificial Intelligence into the lives of programmers with tools like GitHub Copilot.

Copilot and other coding assistants provide programmers with autocomplete for lines and blocks of code, the ability to autonomously analyze and debug complex programs, and a chatbox for requesting fully-written functions, files, or programs. With Copilot, anyone can, in theory, sit in front of a computer and code up a tool, video game, or website. Those who already have coding experience are given a built-in collaborator, ready to suggest improvements and instantly track down their persistent bugs. As a result, the traditional programmer is expected to be "ten times more productive," as OpenAI CEO Sam Altman suggested in a recent interview. This has given rise to the practice of "vibe coding," wherein the programmer simply describes what they want to build and the AI does the rest.

In the paradigm of pair programming, a "driver" is responsible for typing on the keyboard and clicking around the directory, while a "navigator" reviews code and offers guidance. Coding assistants offer to play the role of either driver or navigator, effectively making a two-person operation into an individual activity. The spirit of pair programming is embodied in the Pull Request process, which is initiated whenever a contributor wants to add their code changes to a larger collaborative project. During a PR, the contributor shares their code with a project manager for review, and the manager responds with specific feedback, proposing areas

[4] Roller, Joshua. "Coding From 1849 to 2022: A Guide to The Timeline of Programming Languages." IEEE Computer Society, June 10, 2022. https://www.computer.org/publications/tech-news/insider-membership-news/timeline-of-programming-languages/.

for improvement. Copilot also promises to be a replacement for this point of interaction, as it can do the work of reviewing code, fixing mistakes, and merging changes with the main project branch. Gone are the days of repeatedly alternating between updating and critiquing code contributions. Ironically, it was GitHub, the inventor of this collaborative PR process, that is overseeing its demise with Copilot. All of these new capabilities and expectations constitute the script of the AI coding assistant. Yet, the reality of what the technology can do and how it fits into the experience of open source programmers is more complicated.

Shortcomings and Incompatibility with Open Source

The open-source community emerged from a tradition of collaborative, non-commercial software development and has grown into a vast, decentralized ecosystem driven by volunteer contributors, shared responsibility, and flexible governance. Participation in open-source projects is driven by a "do-ocracy," wherein the most collaborative and consistent contributors are rewarded with greater decision-making power. In the process of engaging with a project, individual developers learn from PR feedback and retain the freedom to choose how much they are invested in their work, according to their passions and community preferences. This incentive structure challenges the script of tools like Copilot, which aim to eliminate the need for collaborative coding while heightening programmers' output.

Before analyzing the possibility and impact of replacing human collaborators in the open source development cycle with AI coding assistants, we must first question whether these assistants can live up to their advertised capabilities. "Vibe coding" has not yet reached the point where a programmer could "[give] into the vibes, embrace exponentials, and forget that code

even exists," as its creator Andrej Karpathy suggested.[5] The reality is that AI-generated code is unpredictable. It often excels at syntax but overlooks important aspects of a project, such as structure, maintainability, and efficiency. Meanwhile, it can introduce invisible bugs and, given the rapid pace of AI development, its expected output is constantly changing. These aspects make it harder for developers to understand and fix problems in their code, which demands more time and effort. The result is that individual programmers are expected to contribute more, especially by large companies that rely on unpaid programmers to maintain popular open-source projects, while taking on the extra burden of fixing faulty code and adapting to constantly shifting technology. This speedup of work paired with labor-intensifying tools mirrors the experience of workers on the factory floor during the mid-20th century, whose labor acceleration was justified by new automotive manufacturing machines.[6]

Even assuming the perfect operation of Copilot and other coding assistants, such that they could effectively act as an productivity booster and code reviewer, this vision of AI-assisted coding does not clearly align with the values of open source development. Those who dedicate themselves to open source development are motivated by the opportunity to learn from one another, not to be an individualized coding machine. When pair programming and PRs are automated away, the sharing of ideas is also lost. Contributors can no longer benefit from the cycle of code review and improvement that has always allowed them to build their skills and discuss high-level project plans with one another. The assumption of speed over struggle simply doesn't align with the open-source approach to coding.

---

[5] Karpathy, Andrej. "Vibe Coding Tweet." X, February 2, 2025. https://x.com/karpathy/status/1886192184808149383.

[6] Resnikoff, Jason. "Labor's End: How the Promise of Automation Degraded Work ." The American Historical Review 129, no. 2 (June 1, 2024): 27–29. https://doi.org/10.1093/ahr/rhae044.

Furthermore, vibe coding threatens the do-ocracy that underpins open source development. When the line between code written by a person and code generated by Copilot is blurred, it becomes unclear how to recognize top contributors and grant them greater project ownership. This is especially true if the AI is also reviewing the code, as the interaction between contributors and maintainers has been completely discarded in favor of efficiency. To preserve this incentive structure, the open source community must consider how it can build AI coding assistants around these crucial connections, not vice versa.

De-Scription and Developer Resistance

In resisting their work's speedup and siloed nature, open source developers will need to find uses for coding assistants that favor collaboration, clarity, and critique over rapid production. This “de-scription” of the technology to fit the unique characteristics of open source development will require programmers to resist the assumptions built into tools like Copilot.

Here, we can look to the example of delivery drivers in the UK, who used the inherent qualities of the algorithmic technology that organized and surveilled them to resist the same technology’s intent to separate them. Specifically, these drivers saw the fact that the delivery app would connect them at algorithmically determined meeting points as an opportunity to share contact information and build an organized network of resistance.[7] This strategy leveraged inherent assumptions about how the app would work to defy its built-in restrictions on driver communication.

[7] Woodcock, Jamie. “Towards a Digital Workerism: Workers’ Inquiry, Methods, and Technologies.” NanoEthics 15, no. 1 (April 2021): 87–98. https://doi.org/10.1007/s11569-021-00384-w.

Much like the drivers' system assumed a need for driver rendezvous, today's coding assistants assume a degree of failure due to their purely statistical nature and limited context window. Their tendency to overlook crucial components of the code's overall structure provides a fascinating site for critique and refinement. Herein lies an opportunity to revitalize the collaborative review process that is so integral to open source projects. By clearly distinguishing between human-written and AI-generated code, whether through in-line comments or GitHub annotations, Pull Requests can once again generate meaningful discussion between contributors and maintainers. Developers can debate why the AI made the decisions it did and contrast these decisions with their own. In this way, the shortcomings of Copilot's output could be a learning opportunity and a chance to demystify the rapidly changing AI ecosystem, while foregrounding collaboration to improve code and outline best practices, regardless of authorship. Precisely by exposing the seams in an interaction that is intended to be seamless, we can discover new moments for constructive community interaction. Even in a distant future where coding assistants have far outpaced our abilities, effectively making us the assistants, this approach of discussing stark differences between our code and the AI's could prove to be invaluable for understanding this black-boxed technology and benefiting from community engagement.

There's also the emerging friction around merit incentives for project participation. Copilot complicates the relationship between effort and recognition. If code can be generated with minimal input, what does it mean to be a "top contributor"? Developers may begin to value different signals, such as engagement with others, thoughtful commenting, or efforts to teach and onboard new contributors. By spotlighting these community-oriented contributions, it becomes impossible to separate the incentive structure and the collaborative components of open source

development. Meanwhile, the expectations of higher productivity and output are sidelined in favor of more constructive values.

De-scription in this context is not necessarily a matter of rejecting AI and the coding assistants it has given rise to. Rather, it's about preserving the social and interpretive labor of open source development; labor that AI coding assistants, by design, tend to obscure. In this resistance, developers have the potential to demonstrate that, while Copilot may script them as hyper-productive individualized workers, they understand themselves as part of a collective project. That collective, in response to the friction between its values and the values of artificially intelligent programming, has the potential to rewrite the script on its own terms.

Works Cited

Karpathy, Andrej. “Vibe Coding Tweet.” X, February 2, 2025. https://x.com/karpathy/status/1886192184808149383.

Lingel, Jessa. “‘Alexa, Tell Me about Your Mother’: The History of the Secretary and the End of Secrecy.” Catalyst, 2020. https://doi.org/10.28968/cftt.v6i1.29949.

Madeleine, Akrich. “The De-Scription of Technical Objects.” Shaping Technology/Building Society: Studies in Sociotechnical Change, MIT Press, 1992.

Mayya, Varun. “Sam Altman On Miyazaki’s Thoughts on Art, Design Jobs, Indian AI, Is Prompt Engineering A Job?” YouTube. Accessed May 1, 2025. https://www.youtube.com/watch?si=s14PKRuVFRBT5GBx&v=xFvlUVkMPJY&feature=youtu.be.

Oxford English Dictionary, “robot (n.1), Etymology,” July 2023, https://doi.org/10.1093/OED/4915451935.

Resnikoff, Jason. “Labor’s End: How the Promise of Automation Degraded Work .” The American Historical Review 129, no. 2 (June 1, 2024): 27–29. https://doi.org/10.1093/ahr/rhae044.

Roller, Joshua. “Coding From 1849 to 2022: A Guide to The Timeline of Programming Languages.” IEEE Computer Society, June 10, 2022. https://www.computer.org/publications/tech-news/insider-membership-news/timeline-of-programming-languages/.

Woodcock, Jamie. “Towards a Digital Workerism: Workers’ Inquiry, Methods, and Technologies.” NanoEthics 15, no. 1 (April 2021): 87–98. https://doi.org/10.1007/s11569-021-00384-w.