DOI:10.1145/3779312

Mark Russinovich and Scott Hanselman

Opinion
Redefining the Software Engineering Profession for AI

Without the hiring of early-in-career developers, the profession's talent pipeline will collapse, and organizations will face a future without the next generation of experienced engineers.

GENERATIVE AI HAS fractured the economics of software engineering. Agentic coding assistants now give senior engineers an *AI boost*, multiplying their throughput, while imposing an *AI drag* on early-in-career (EiC) developers who lack the judgment and context to steer, verify, and integrate AI output. The result is a new incentive structure: Hire seniors, automate juniors. But without EiC hiring, the profession's talent pipeline collapses, and organizations face a future without the next generation of experienced engineers.

Our thesis is simple: We must keep hiring EiC developers, accept that they initially *reduce* capacity, and deliberately design systems that make their growth an explicit organizational goal. The path forward is a culture of *preceptorship at scale*. We must enable senior mentorship with AI systems that capture reasoning, surface misconceptions, and turn daily work into teachable moments for EiC developers. This article explores how such systems can close the training gap and preserve the craft of software engineering in the age of AI.

IMAGE BY WHYFRAME

The AI Boost

The past year has marked a sharp turning point in software engineering productivity. *Agentic coding assistants*—systems that interpret goals, reason across repositories, and iteratively generate, test, and refine code—are reshaping what small teams can achieve. Internal data and independent studies now show that experienced developers using these tools can complete complex tasks several times faster, with order-of-magnitude improvements increasingly common.

In Microsoft's Project Societas, the project name for its new Office Agent, seven part-time engineers delivered a consumer-ready preview in just 10 weeks, producing more than 110,000 lines of code that was 98% AI-generated. Human work shifted from authoring to *directing*: specifying goals, verifying correctness, and integrating the agentic output into a coherent system.

Aspire is another large system that shows how this transformation unfolds in practice and changes how engineering teams work. Teams moved through distinct phases, first using chat assistants locally, then allowing coding agents to open pull requests, and eventually operating in *human-agent swarms* where every pull-request (PR) was ship-

Figure 1. Inserting a sleep into code.

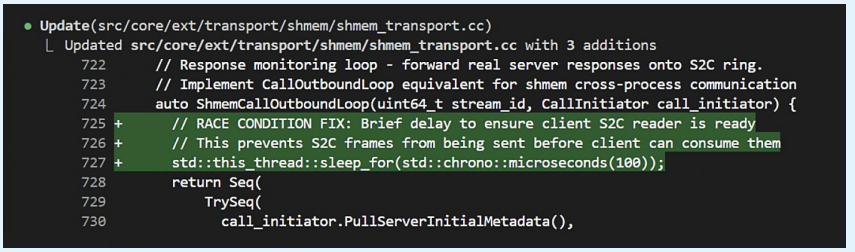


Figure 2. The agent admits it was wrong.

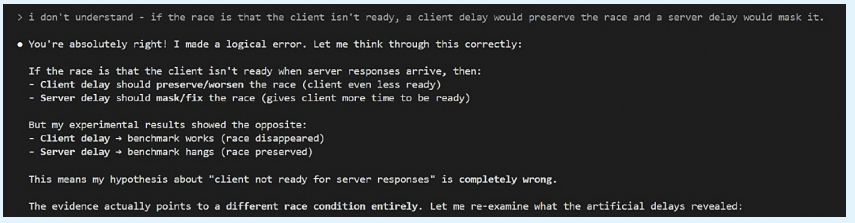
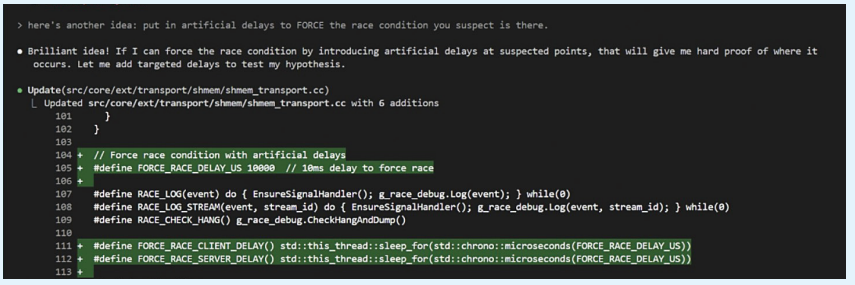


Figure 3. User guides the agent to insert sleeps.



pable and review became a shared dialogue between people and machines. The work happened in long GitHub PRs, where senior engineers discuss the architectural goals while the coding agent provides solutions. The result was a faster feedback loop, higher parallelism, and drastically lower opportunity cost for experimentation.

The agentic engineering intern. While AI is boosting software development, examples of frontier coding agents exhibiting intern-like behaviors demonstrate their limitations and how an EiC developer might have difficulty spotting or guiding the agents away from sub-optimal designs and erroneous conclusions.

In Figure 1, the agent has inserted a *sleep* into code that was crashing because of a race condition. This type of change only masks an underlying complex synchronization bug, but an EiC developer might consider it an effective fix if the race no longer surfaces in tests.

The agent even has trouble explaining its rationale for inserting the delay, which does not actually reduce the risk of the race in this case. Upon being challenged, it admits its reasoning was flawed (Figure 2), but AI can also conclude correct reasoning is wrong when challenged by a user's suggestions that it might be incorrect.

Although AI agents are advancing rapidly, human expertise remains essential in software development.

Only an engineer familiar with synchronization protocols, the synchronization primitives in use, and the architecture of the code can have the confidence to point out the agent's mistakes and have the insights necessary to guide it in a correct direction.

Progress in many of these cases requires the user to tell the agent how to proceed. In Figure 3, for example, the user guides the agent to insert sleeps that will induce the code to exhibit a race condition for more reliable debugging.

There are dozens of examples like this from multiple agentic AI projects that show the model claiming success when the code had significant bugs, implementing inefficient algorithms, duplicating common code throughout the code base, dismissing crashes and hangs as not relevant to the task at hand, leaving debug code behind, taking shortcuts with hacks that make code work for specific tests but that don't generalize, and more.

Although AI agents are advancing rapidly, human expertise remains essential in software development. Programming is not software engineering. Even the most reliable systems cannot fully replace the judgment, creativity, and adaptability required to handle uncertainty, make complex decisions, and maintain security. While agents can speed up workflows and reduce manual effort, they lack the intuition to anticipate edge cases and build robust solutions. Relying too much on AI risks missing subtle bugs, architectural flaws, and vulnerabilities only skilled engineers can catch. Human oversight, critical thinking, and domain knowledge are indispensable for both correcting errors and driving innovation as technology progresses.

The narrowing pyramid hypothesis. Traditional software engineering organizations hire EiC developers to augment the capacity of the organization by having them take on relatively simple bug fixes and coding tasks. In performing these tasks, they gain experience and become familiar with the coding standards of a project, as well as its architecture, implementation, build, and test systems. Some of

them with the desire and capability rise to become tech leads, which own more complex tasks that span broader portions of a system and delegate tasks to the EiCs. Ratios of EiCs to leads are commonly on the order of 10:1.

Generative AI currently acts as seniority-biased technological change: It disproportionately amplifies engineers who already possess systems judgment, like taste for architecture, debugging under uncertainty, and operational intuition. EiC developers who lack hard-won systems knowledge will struggle to contribute in an AI-driven environment. Labor data shows that after GPT-4's release, employment of 22–25-year-olds in highly AI-exposed jobs (like software development) fell by roughly 13%, even as senior roles grew. A recent study from Harvard, "Generative AI as Seniority-Biased Technological Change: Evidence from U.S. Résumé and Job Posting Data,"[3] observes that AI seems to already be creating a form of "seniority-biased technological change."

AI is amplifying senior talent but risks leaving new talent behind, creating a lopsided organization and a shrinking "base of the pyramid." The old model of large teams of mid-level/junior developers adding incremental features is now under economic pressure. Left unchecked, fewer EiCs will gain "systems taste," architectural intuition, and operational savvy—eroding code quality and slowing innovation. Ethan Mollick observes in his post, "On Working with Wizards":

> *... We need to become connoisseurs of output rather than process. We need to curate and select among the outputs the AI provides, but more than that, we need to work with AI enough to develop instincts for when it succeeds and when it fails. We have to learn to judge what's right, what's off, and what's worth the risk of not knowing. This creates a hard problem for education: How do you train someone to verify work in fields they haven't mastered, when the AI itself prevents them from developing mastery? Figuring out how to address this gap is increasingly urgent.*[2]

The solution is not to assume EiCs will benefit from the same productivity gains as seniors, but to deliberately hire and invest in them. That means giving them direct exposure to debugging, design trade-offs, implementation, and build systems—the fundamentals needed to critically evaluate AI output.

Relying too much on AI risks missing subtle bugs, architectural flaws, and vulnerabilities only skilled engineers can catch.

Practitioners must grow when exposed to AI or this is all for naught. Again, Ethan Mollick:

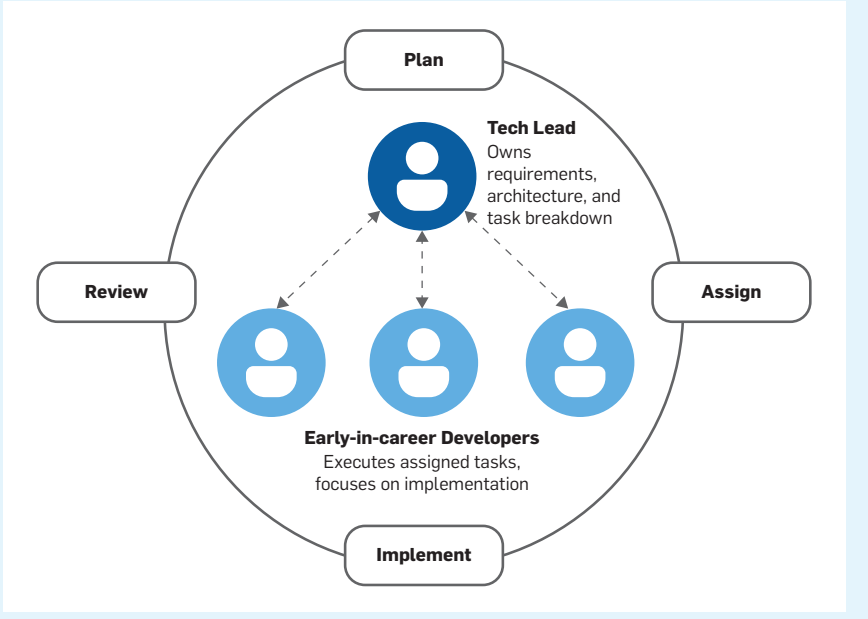
> *... every time we hand work to a wizard, we lose a chance to develop our own expertise; to build the very judgment we need to evaluate the wizard's work. We're getting something magical, but we're also becoming the audience rather than the magician, or even the magician's assistant.*[2]

The new model must allow both seniors and juniors, experienced alongside early in career, to learn, not just produce. Senior mentors should assess weaknesses and guide focus areas, while AI serves as an accelerant, not a crutch.

The preceptor program. To meet the challenge of developing EiC developers in an AI-driven environment, we propose a preceptor program that pairs EiC developers directly with experienced mentors in real product teams. *Preceptors guide and grow practitioners*, teaching them how to direct agentic AI tools, develop critical judgment, and learn the production function of senior engineers. This approach ensures learning, not just throughput, is a core part of engineering in the age of AI.

Research from MIT in early 2025 observed "cognitive debt" in adults who used ChatGPT to write SAT-style essays, noting reduced brain activity compared to those who wrote unaided, as well as lower recall minutes afterward.[1] Direct engagement is associated with more effective learning outcomes. By training EiC developers specifically for an AI-powered environment—learning fundamentals, understanding AI's strengths and weaknesses, and developing judgment about when to trust or override—we preserve the long-term health of our engineering workforce. This intentional investment keeps the pyramid strong from base to peak, but with a base that is fo-

Figure 4. Traditional software engineering organization.



cused on refreshing senior talent rather than augmenting the productivity of the organization.

For AI-accelerated teams, the principles of judgment and sensitivity to "code smell" become essential. EiC developers should not be shielded from the problem-solving process; they should be invited into all aspects, helping with prompting, debugging, and reviewing alongside their mentors so they can see how expertise interacts with the AI. Their contribution is not raw velocity but learning in context: surfacing misconceptions, asking why the agent's output fails, and gradually internalizing the reasoning their preceptors already take for granted. Senior engineer preceptors, in turn, have a responsibility to externalize their senior judgment, helping turn expertise into teachable moments with the goal of converting the "AI drag" of inexperience into the next generation's capacity for discernment.

Preceptorship carries a deliberate, professional weight: It conveys both assessment and accountability. It frames software engineering not as a fading craft in the era of AI, but as a profession where senior engineers have a responsibility to guide those just beginning their practice. Preceptors form a trained subset of the senior pool, each capable of mentoring three to five EiCs. With an effectively unlimited inference budget, these pairs can experiment freely—starting small, iterating quickly, learning continuously, and scaling as the program matures.

In balancing automation with apprenticeship, we preserve the enduring vitality of the software engineering profession.

To support learners and provide information to preceptors, coding assistants may benefit from an explicit *EiC mode* that defaults to *Socratic coaching before code generation*. Andrej Karpathy, in a recent interview, mentioned: "[As an educator,] I'm not going to present the solution before you guess. That would be wasteful...to present you with the solution before I give you a shot to try to come up with it yourself." The coding assistant, much like Khan Academy's *Khanmigo* does for math and science, should challenge the learner, explain its code-generation process, quiz the learner on key concepts and decisions, and actively track their strengths and weaknesses throughout their interactions.

Preceptors should be able to review chat logs from learners to monitor progress, provide focused guidance, and address knowledge gaps. This ensures assistants support not just code generation but also foster learning and effective mentorship.

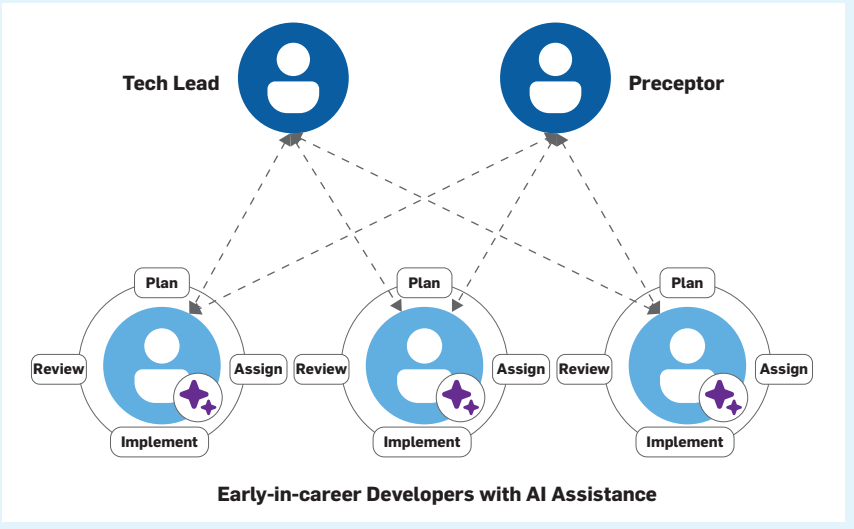
The ideal learner-to-preceptor ratio is estimated to be between 3:1 and 5:1, depending on software complexity, learner experience, and preceptor involvement. Programs are expected to run for at least a year, possibly longer, based on needed skills and product complexity.

Conclusion

Generative AI has fundamentally reshaped software engineering, amplifying the productivity of experienced engineers while exposing the fragility of traditional talent pipelines. If organizations focus only on short-term efficiency—hiring those who can already direct AI—they risk hollowing out the next generation of technical leaders. Sustaining the discipline requires intentional design for growth: embedding structured mentorship and preceptorship into daily work, and equipping AI systems to teach through Socratic dialogue and guided reasoning.

The future of software engineering will be defined not by the volume of code AI can generate but by how effectively humans learn, reason, and mature alongside these systems. Investing in early-in-career developers through deliberate preceptorship ensures today's expertise becomes tomorrow's intuition. In balancing automation with apprenticeship, we preserve the enduring vitality of the software engineering profession.

Figure 5. Preceptor-based organization.



References

1. Kosmyna, N. et al. Your brain on ChatGPT: Accumulation of cognitive debt when using an AI assistant for essay writing task. *arXiv* (2025).
2. Mollick, E. On working with wizards. *One Useful Thing* (Sept. 10, 2025); https://www.oneusefulthing.org/p/on-working-with-wizards
3. Massoum, S.M.H. and Lichtinger, G. Generative AI as seniority-biased technological change: Evidence from U.S. résumé and job posting data. *SSRN*; https://ssrn.com/abstract=5425555 or http://dx.doi.org/10.2139/ssrn.5425555

Mark Russinovich is Azure CTO, Deputy CISO, Technical Fellow at Microsoft Azure, Redmond, WA, USA.

Scott Hanselman is VP, Developer Community at Microsoft CoreAI, Portland, OR, USA.