SDS PODCAST

EPISODE 970: THE "100X ENGINEER": HOW TO BE ONE, BUT SHOULD YOU?

Jon Krohn:	00:00	This is episode number 970 on the 100 X engineer. Welcome back to the Super Data Science Podcast. I'm your host Joh Krohn. Two weeks ago in episode number 966, I had an episode focused on open claw, which has created a lot of hype around getting agentic support on a wide range of tasks, including coding. Then a week ago in episode 9 6 8, I had an episode focused on how while code generation tools have made writing code a lot faster and easier, it seems to be creating more coding jobs than it's taking away. But how practically can we leverage code generating models to be vastly more productive ourselves? Well, that brings us to today's topic of the 100 x engineer. Throughout my career, I've heard of 10 x engineers and 10 x data scientists, and I dare say I've even worked with some folks whom you could consider to be the latter.
	00:55	But now because of the rapidly increasing capability of code gen tools like Anthropics Cloud Code and open AI's Codex, people are starting to talk about 100 x engineers, which in my view includes any kind of technical role where coding is a primary function. I should add a disclaimer that Anthropic recently became a sponsor of this show. So if you're listening to this episode shortly after its release, there's a good chance a Claude Code sponsor message will be digitally inserted into it. So thanks to Anthropic for supporting the show. We couldn't make it without sponsors like them, but I already have this episode planned before Anthropic came to us about their campaign, and so everything positive I say in this episode is based on my genuine enthusiasm for and personal enjoyment of these powerful and very often even fun to use tools and this is no infomercial.
	01:41	I also address limitations and some of the problems that could arise if you yourself attempt to 100 x your code

output to set the stage. Lemme start with a social media post that's been making the rounds in late January. Andrej Cari, who many of you will know as one of the original co-founders of OpenAI and a former director of AI at Tesla, he tweeted what he called a few random notes from Claude Coating, and these few random notes have racked up over 7 million views. Karpathy described how in just a matter of weeks around December, he went from about 80% manual coding with auto complete assistance and 20% AI agents to the inverse. He now does 80% AI agent coding with only 20% manual edits and touch-ups as car Pathi put it. He's now mostly programming in English telling the LLM what code to write in words.

02:29 He admitted that it hurts his ego a bit to do that, but the power to operate over software in large code actions is simply too useful to go back. Now, Karpathy is obviously an elite engineer, so you might wonder how representative his experience is, but he noted that he'd expect something similar is happening for well into double digit, well into double digit percentages of engineers out there, even while awareness of the shift in the general population is still in the low single digits. That gap between what coders are experiencing day to day and what people realize is happening is I think quite significant, and I talked about that a fair bit in last week's episode, particularly as it pertains to that blog post Something big is happening, which listen to last Friday's episode or check out that blog post for more on that gap, and I also recommend reading Car Path's tweet in full.

03:21 I've of course got it for you in the show notes, but to summarize his key points, he was candid about the limitations of code gen tools too. The AI agents don't make simple syntax errors anymore. The mistakes have evolved into subtle conceptual errors. The kind that a hasty junior developer might make, the most common category he said is that models make wrong assumptions on your

behalf and charge ahead without checking. They also may overcomplicate things. They can produce a bloated, brittle construction over a thousand lines of code, and it's up to you to say, could you just do this this other easier way instead, at which point it'll cut down those thousands of lines of code to say a hundred lines despite these issues, which in my view will no doubt improve rapidly over the coming months. Karpathy was unequivocal. He said Code generation agents are a massive net improvement and it's very difficult to imagine going back.

04:12 I'm sure if you like me, have experienced these tools, you feel like you're in the same boat. It really is fun, but here's where things get really interesting and where the title of today's episode comes from. There's a developer named Peter Steinberger who has essentially become the poster child for what this new paradigm looks like when somebody goes all in. Steinberger story was profiled in a viral LinkedIn post by an AI engineer named Brian Vazquez a couple of weeks ago. I've of course got a link to that for you in the show notes as well, and the data are staggering. Over a two month period, Steinberger racked up over 6,500 commits on GitHub that works out to roughly 210 commits per day on average, and in that same two month, he added 2.5 million lines of code and removed 1.9 million lines of code. To put that in perspective, many engineering teams ship a few hundred commits per month, and he was doing that on average every day for two months.

05:10 And here's what makes this story even more compelling. Steinberger isn't some fresh graduate running on caffeine. He's a 40 something founder who built a successful PDF framework called Nutrient formerly P-S-P-D-F kit used by companies like Dropbox, Autodesk and IBM. He bootstrapped that company or that product for a decade and sold it in 2021 for a reported hundred million euros, and then he burned out so badly that he

quit coding entirely. Took a three-year sabbatical and many founders in that situation never come back to the keyboard. But last year, or sorry, in 2024, Steinberger started experimenting with AI coding tools and completely rebuilt his relationship with software development from the ground up. So what does his workflow actually look like? It's simpler than you might expect, a single large curved monitor with a three by three grid of terminal windows, each one running an AI coding agent.

05:59 He has three to eight agents running simultaneously, each working independently for five to 15 minutes, and he rotates his attention across them like a conductor directing an orchestra reminding me of Sadie St. Lawrence's book becoming an AI orchestrator, which you could hear about from her directly back in episode 955. Anyway, Steinberger commits directly to the main branch with no pull request overhead. The tooling itself isn't exotic, it's the parallel execution at scale that matters, but the real magic happens before Steinberger even opens those terminals. He uses a voice first specification system. He dictates raw, unstructured ideas using voice to text. Software then uses AI to convert that stream of consciousness into a structured software design document. Subsequently, and this is the clever part, he copies that spec into a fresh AI context and asks it to tear the specification apart, identifying 20 points that are underspecified weird or inconsistent.

06:53 He iterates on this adversarial review until the questions become increasingly niche, which is the signal that the spec is solid. The result is highly detailed specifications often over 500 lines long, so thorough that implementation becomes almost trivial. He drops the spec into cloud code with a single prompt and lets the agents build. The counterintuitive insight here is that Steinberger spends more time on specifications now, not less. Instead of the traditional ratio of roughly 20%

planning and 80% coding, he's closer to 60% planning and 40% AI execution. This connects directly to something car Pathi observed in his tweet. The best way to get leverage from AI agents is to shift from an imperative approach, telling the model exactly what to do step by step to a declarative one, giving its success criteria and letting it figure out the path. Karpathy recommended writing tests first and then having the agent pass those tests or writing a naive algorithm that's very likely correct, and then asking the agent to optimize it.

07:49 In both cases, you're specifying what you want, not how to get there. Carpathia also made an insightful observation about the nature of the speedup. It's not just that he's faster at what he was already going to do. The main effect is that he does a lot more than he otherwise would've because he can now code up things that simply wouldn't have been worth the effort before, and he approaches code bases that he couldn't have worked on due to knowledge or skill gaps. So it's less of a pure speed up and more of an expansion of what's possible. More on that later in my recommendations of what you might want to do here, because both Karpathy and Steinberger are upfront about downsides they experience from the way that they're using cogen tools. Karpathy, for example, noted, he's starting to feel his ability to write code manually.

08:33 Atrophying generation and discrimination, as he put it, are different capabilities in the brain and some folks can review code well even if they struggle to write it from scratch. Steinberger for his part has called himself a CLA aholic and says he's addicted to age agentic engineering well, so maybe desiring to become a 100 x engineer can mean going a bit too far with what code gen tools allow you to do. There's also a broader question about quality and accountability that I think is important to flag if you

want to become a hundred x engineer. Steinberger has been quoted saying that he ships Cody never reads, which understandably raises eyebrows Steinberg's answer to the obvious question, to the obvious concern, sorry, was already suggested by Karpathy earlier in this episode, and that's tests. If the test pass, the implementation is correct regardless of how the code looks, that's a defensible position, but the number of lines of code committed is certainly not the best benchmark of quality.

09:26 Perhaps it's better to aim then to become merely a 10 x or five x or two x as productive as you are without code gen tools while keeping a close eye on code quality personally or maybe don't aim to be any more productive at all. Instead, aim to be able to tackle a wider range of coding problems with your work than you ever could before. Understanding and learning from these tools as you progress. Looking beyond what's possible today or how you might best leverage code gen tools for your particular use cases. I'd like to wrap up this episode with some thought provoking questions sparked by car paths. Tweet what for example, will happen to the ratio of productivity between the average engineer and the best engineer. It's quite possible that that gap grows enormously with rarer folks like Steinberger vastly outpacing the rest of us, or will the productivity gap shrink because accessibility and ease of use continue to improve?

10:18 Will generalists at some point outperform specialists since LLMs today at least are better at filling in the micro details than they are at grand strategy and my personal favorite of Karpathy, these questions. What does LLM assisted coding feel like in the future? Is it like playing a video game like StarCraft or is it so immersive rich and rhythmic that it's like playing music that it really does feel like conducting an orchestra? Who knows where this human machine symbiosis will take us exactly, but it'll

definitely be interesting and continue to be fast evolving thanks to the length of human coding tasks that Frontier LLMs can handle doubling every seven months. Where does all of this leave us? Caries assessment is that LLM agent capabilities cross some kind of threshold of coherence around December and caused a phase shift in software engineering. The intelligence part of code generation, that's the understanding of our input tokens and the quality of the output tokens suddenly feels quite a bit ahead of everything else.

11:18 The integrations, the organizational workflows, the broader diffusion across the industry, and steinberg's extraordinary output, whether you view it as inspiring or cautionary or both demonstrates what's possible when a deeply experienced engineer fully embraces these tools with zero attachment to how things used to be done. These are exciting and frankly disorienting times for anyone who builds software for a living who writes code for a living like I assume a lot of you listeners do, but I do think the takeaway is clear. The coders who thrive in 2026 and beyond will be the ones who learn to think declaratively, who invest heavily in specifications and testing and treat AI agents not as replacements for their expertise, but as extraordinary amplifiers of it, dream up something big and go build it. There's never been an opportunity like this one, and if you're looking for a place to start, this is not a paid promotional message.

12:08 But my wonderful friend Ed Donner released a brand new 16 hour Udemy course, which I've got for you in the show notes on how to go from vibe coding to being an agentic engineer through leveraging code generation tools, exactly like we've been talking about in this episode. It covers cloud code copilot Codex cursor, antigravity, MCP, and lots more to make you more productive, accurate, and successful with code gen tools. I've only just begun the course, but I'm loving it so far. All of his early students

seem to be loving it as well because it has an average 4.9 star rating. If you aren't aware of Ed Donner already, he's a megastar of a agentic AI training with over 400,000 paying students, 400,000 of them on Udemy. This his latest course is different because all of his previous ones were about writing code to engineer AI agents.

12:54 While this one you are leveraging AI agents to write code could be the perfect place to start if you're looking to wrap your head around all the possibilities and accelerations available at this time like I am. Alright, that's it. If you enjoyed today's episode and know someone who might consider sharing this episode with them, leave a review of the show on your favorite podcasting platform or YouTube, tag me in a LinkedIn post with your thoughts, and if you aren't already, obviously subscribe to the show. Most importantly, however, I just hope you'll keep on listening. Until next time, keep on rocking it out there and I'm looking forward to enjoying another round of the SuperDataScience Podcast with you very soon.