3 Types of AI-Assisted Programmers

2023-11-27T20:21:14-07:00

Large Language Models (like ChatGPT) write very convincing code snippets. Early in 2023, this discovery caused a lot of premature celebration among LinkedIn influencers: no longer was there any need for seasoned developers with their gatekept expertise, fastidious attitudes, and inconveniently high salaries. Now anyone with an Internet connection could ask an AI model to write the next big social media app, in JavaScript please, and throw in some blockchain while you’re at it.

The promised wave of apps built entirely by AI never materialized. I’m trying my best not to gloat about it.

Still, there’s a compelling case for AI as a programming tool. Code is a computer-oriented language. It has a small, unambiguous vocabulary and unbreakable syntax rules. Patterns and repetition are its bread and butter. It’s predictable to a fault, which is why computers (and some humans, like myself) are so compatible with it. If anyone could write good code, it would be a computer. And if generative AI’s greatest strength is its ability to model and imitate patterns, forming a fluid interface between humans and machines—[as I’ve previously argued](https://stackoverflow.blog/2023/05/01/ai-isnt-the-app-its-the-ui/)—wouldn’t programming be the ideal use case for it?

Well, yes and no. AI-powered programming tools have made a splash in the programming world, and they’re probably not leaving anytime soon. But despite having arrived for junior, mid-level, and senior engineers all at once, not to mention people who don’t know a struct from a hole in the wall, the risk/benefit calculation couldn’t be more different depending on your level of experience. AI could be the tool that fast-tracks your career or it could be the obstacle that derails it. What matters isn’t just *whether* you use it, but *how*.

Let’s take a look at what AI means for developers (and non-developers) at every stage.

## The no-code entrepreneur

Many people’s first taste of programming came this year in the form of a ChatGPT conversation. It’s a seductive experience: you can ask it to write an application in any major programming language and it will spit out code right up to the token limit—more than enough space for the typical “tutorial-sized” app. From there, you can ask for tweaks and bugfixes until you’re satisfied with the output. And when you paste it into an IDE and it actually *works*, it feels like you’ve cracked the industry wide open.

The reason for this rosy first impression is that the problems are hidden under the surface. AI tools are trained on code from thousands of real projects with disparate levels of quality and completeness—that is, code that almost always runs and usually gets the job done, but is only occasionally reliable, maintainable, secure, or bug-free.

Studies have found that tools like ChatGPT, GitHub Copilot, and Amazon CodeWhisperer deliver code that is “valid” (runs without errors) [about 90% of the time](https://arxiv.org/pdf/2304.10778.pdf), passes an average of 30% to 65% of unit tests, and is “secure” [about 60% of the time](https://arxiv.org/pdf/2108.09293.pdf). Note that these studies rely on well-written prompts created by engineers. Insignificant changes to the wording of a prompt can result in [significant differences](https://arxiv.org/pdf/2302.00438.pdf) to the code output. And they only test the AI’s ability to output “snippets”—small pieces of straightforward code. There is no data on AI’s ability to write applications as complex as the average legacy app. It’s probably unable to do so at all, just as ChatGPT is unable to write a coherent novel.

So where does AI fit for people who can’t code?

Some would say it doesn’t. The idea of AI-dependent programming ruffles a lot of feathers in the software community. A programmer is someone who knows how to code. How can you call yourself a programmer if you can’t even write an if statement? But this misses the point. As only a programmer would need to be reminded, if statements (like all programming logic) aren’t an asset, they’re a liability. The best code is [no code at all](https://github.com/kelseyhightower/nocode), and second best is the minimum amount of code that solves the user’s problem. If it were possible to build high-quality apps without writing a single line of code, there wouldn’t be anything wrong with that.

Unfortunately, it isn’t, and probably won’t ever be. Code, as a category, is nothing more and nothing less than being ridiculously specific about what you want. If your AI prompts are detailed enough to produce exactly the right code, you *are* coding in every way that matters. However, AI is non-deterministic; it doesn’t always produce the same output from a given input. There’s an element of randomness. So even prompts that qualify as code are *unpredictable* code, and unpredictability is the last thing you want after spending hours or weeks or months figuring out the minute details of a process. Anyone who spends time coding via AI prompts will eventually come to wish for something more direct, something more structured, something they can rely on to behave the same way at all times—they’ll wish for programming languages and compilers.

I predict AI will become a gateway drug for some future programmers. But there’s another, more important niche it can fill, and it’s one that’s easy to overlook.

There are many situations where software isn’t needed, but code is. Professionals in other fields already use AI to write one-off SQL queries and VBA macros. It could also be used to create app prototypes for pitch decks, proof-of-concept workflows on the command line, or disposable data-scraping bots. If you need software, you’ll have to work with a software professional. But if you just need a bit of short-lived code and are willing to deal with rough edges, there’s nothing wrong with shaking the AI and seeing what falls out.

Ultimately, this can only lead to more work for programmers. As AI bridges the gap between vision and first draft, early-stage software companies will proliferate, and any startup that goes to market will discover (sometimes very urgently) that they need engineers on staff. But for some of them, especially the ones that started out with fewer connections and less money, the fact they’re able to reach that point at all will mean AI has done its job.

## The junior engineer

New programmers have the most to lose—and, in equal measure, the most to gain—from AI tools.

Practically every junior developer feels overwhelmed at their first job. It’s like moving to Spain after a year of Duolingo lessons: it’ll be at least a few months, probably a lot longer, before you have a clue what’s going on. Real-world applications aren’t like the compilers you wrote during your senior year of college or the showcase projects you worked on at programming bootcamp. The depth and complexity are a hundred times greater, and the standards are higher—it can feel impossible to keep track of all the rules that will get you past a senior dev’s PR reviews.

All of this considered, it must be incredibly tempting for junior devs to pull up ChatGPT and see if it can take some of the pressure off. And for a little while they may get away with it. AI tools (as discussed earlier) are pretty good at the bare minimum, which is a lot better than nothing. And more importantly, they never respond with a blank page and a blinking cursor.

But eventually it will be time to pay the piper. This is the greatest risk around AI code tools: developers who rely on them may never *become* good. If a developer habitually uses a code generator and relies on external feedback loops (PR reviews, integration tests, bug reports, etc.) to find problems, they’ll never understand their own code. This will backfire, and it will be embarrassing. There are critical bugs, attack vectors, technical debt, and other problems in every production app that can only be fixed by someone who has a thorough and correct understanding of the code.

The safest route for a junior developer is to stay away from AI tools. But it’s not the only good route. If you use AI to *gain* understanding instead of circumventing it, it doesn’t have to hold you back.

For example, say you’ve just finished writing a function and you’re feeling uncertain about it. You’ve read through it a couple times and fixed some formatting issues, but you still feel like it’s not quite as efficient or idiomatic as it could be. Before you message a teammate or submit a PR, you could use AI to get another perspective: tell the AI what you’re trying to do, ask it to write a function, and compare its code with yours.

Keep in mind that generative AI is, by constraint, as mediocre as possible. You can’t trust the output. The purpose of this exercise isn’t to give you better code for free, it’s to help you critique yourself. Maybe the AI used a standard method you forgot about, and it would help express your intent better. Maybe it was able to iterate your data set with a single loop instead of two nested ones. Maybe it didn’t give you anything interesting or new; in that case, you probably didn’t miss anything too obvious.

You could also use AI to generate examples of syntax or patterns you’re struggling to understand. If you’re confused by null-coalescing operators, you could ask it to generate examples of their use in context. It won’t generate high-quality code, but you need quantity, not quality. Repetition is the key to learning. If you’ve been reading about the [adapter pattern](https://refactoring.guru/design-patterns/adapter) and can’t find an example of what it looks like in Dart, you can ask for a demonstration. The result may only be mediocre but it will be specific, which is what’s valuable here.

Whenever possible, you’ll still want to learn from official documentation, hand-written code, programming blogs, and Stack Overflow answers. There’s always a risk of AI saying something completely wrong. But if a snippet of subpar, made-to-order code would be enough to get you to the next step, AI can be a good resource.

## The senior engineer

As a senior engineer, you won’t feel the same temptation to use AI as a replacement for fundamental skills. You already have those skills. Maybe you won’t feel inclined to use it at all, and that’s completely fine—you’re never wrong for deciding not to use a particular tool.

Some people wonder what purpose generative AI can possibly serve for an experienced dev. Half the time it spits out junk, half the time it writes something serviceable but not as good as you’d write on your own. Why delegate your job to something that’s objectively worse at it? To answer that question, it may help to clarify the boundary between yourself and AI. At its best, AI-written code only has about a 50/50 chance of doing what it’s supposed to, and hardly any chance of accurately expressing its place in the context of an application. You’re right: AI is not “good” at writing code. However, it is very, very fast. That’s the expectation you should have. **It’s the AI’s job to be fast, but it’s your job to be good.**

Editing and refactoring a piece of code is usually (not always) faster than writing it from scratch. I participated in the beta of GitHub Copilot, using it for contract work on my personal computer, and found that it noticeably increased my development speed. It did nothing for the correctness or maintainability of my code, of course. That was never its job. But by giving me something I could use (with substantial revision) about half the time, it saved a lot of keystrokes overall. Saving time isn’t so great if you’re getting paid by the hour, like I was. But if you’re salaried, it can enable you to spend less time writing boilerplate and more time focused on development processes, code quality, documentation, or any of the other things that make your software sustainable.

The pitfall to avoid here is using AI to increase velocity: delivering *more* features instead of better ones. Velocity is an imaginary, unreliable metric even on the best of teams; humans just aren’t that consistent. And AI is even less so. If you allow expectations to form around your development speed with an AI tool, you’ll find it can’t keep the pace. There are some types of tasks where it excels, but others where it can’t code itself out of a paper bag. It’ll usually give you a boost, but often you’ll be left to figure things out on your own.

Again, this is a question of boundaries. AI is all about speed. If you reinvest the time it saves in more speed, instead of higher quality, you’re squandering your own value as a developer. AI knows how to save time. *You* know how to architect and build great applications.

AI can also be helpful when you need to cross into unfamiliar programming territory. At my day job I write SQL, C#, and TypeScript, but on occasion I’ve had to write snippets of Groovy, MDX, or KQL. The spin-up time for an unfamiliar language is at least a couple of days, but it’s all syntax. Like most senior devs, I can recognize good code by heuristic—the language is less important than the structure—but it still takes time to figure out how to declare a constant or iterate an array correctly. For occasions when you’re outside your wheelhouse, AI can get you there faster. A quick generated snippet may be all you need if it’s throwaway code. And if it’s not, you can spend a little time polishing it up and still come out ahead.

## The future of programming with AI

It’s often argued that generative AI is only in its infancy and will improve by leaps and bounds as time goes on. But the burden of proof on that idea is very heavy. According to [one OpenAI engineer](https://nonint.com/2023/06/10/the-it-in-ai-models-is-the-dataset/), LLMs are little more than an approximation of their dataset. And for that dataset—the Internet as a whole—the most we can hope for is that it won’t get worse. With the web’s current incentive structure (SEO, content marketing, spam, and advertising) it seems very unlikely it will get better.

My favorite definition of programming is “teaching computers to make the same mistakes as humans.” Nowhere is this more literally true than in generative AI. It’s trained almost entirely *on* our mistakes. And as long as we keep writing open-source code for it to scrape, Copilot and CodeWhisperer will keep recycling our mistakes, converging toward the point of perfect, Platonic mediocrity. AI will become the most average programmer in the universe, albeit one that types a thousand words per minute.

Some of you wouldn’t hire a mediocre programmer no matter how fast and cheap they are. I have no argument with that. But if a firehose of mid-quality code could fit into your process somewhere, and you’re wary of the pitfalls, AI can be a great tool to have in your team’s toolbelt.