Coding as thinking - What is lost by coding with AI

Written by Seth Corker on Benevolent Bytes

Banner by photo by Austin Distel on Unsplash

We use writing as a tool to help us think, and the same is true for programming. AI is increasingly being used as a replacement for writing, but this often circumvents deep thinking. It's a tool which is used as a crux to forgo structuring ideas and discovering new ones, ultimately making it difficult to improve your craft.

Writing is a tool we use that shapes the way we think, when we're forced to structure our ideas, we find gaps, we pull on threads that lead us in new directions. This is true with writing, it's also true for programming. As you write code, you get a "feel" for the ergonomics of a system and where it falls flat. Using an LLM to build software takes away the opportunity to organise your thoughts, solidify your ideas and iterate meaningfully on a problem. You're giving up the opportunity to develop thoughts and gain an understanding of the domain when you give that responsibility to AI. I see programming is a tool of communication. The code you write now is a tool to be interpreted by a machine but also something that communicates across time, it's something that future programmers will read and need to understand. Code is also deep thinking distilled. It's the minimum amount of text we get into an editor to communicate the idea and constraints of what we're attempting to model. LLMs lack this same level of understanding. LLMs lack nuance. They're not creating something with a rich knowledge of the problem, they're providing the average conclusion to a prompt. Will you know why an LLM solved a problem in a certain way? Can you communicate that to a colleague in 2 years time?

Proponents of LLMs and using AI tools in programming often point to it being an accelerator because it removes the drudgery and mechanical nature of writing boilerplate. There are definitely some things you write over and over again as a software engineer, but it's been a long time since I remember writing large amounts of what I would consider "boilerplate" code. Small snippets, yes. Large swathes of code, not really. Not since the days of Java. The majority of what I write is contextually dependent on the problem and the surrounding code. I struggle to see how LLMs can have sufficient context to actually make good changes to a codebase of sufficient scale. From what I've seen, they excel at creating new code. New projects where the goal has been written about in blog posts across the internet, fleshed out in countless tutorials and docs. Writing code is not always a mechanical endeavour which can be automated away, especially without the context of the codebase and constraints we hold in our heads of the real-world problems.

Continually prompting and re-prompting an LLM to get the desired output robs you of thinking critically about your code. If you're only concerned with the output, you miss the journey it took to get there, the learnings you had on the way, the challenges to overcome and improve. Delegating coding tasks to AI means you're unlikely to improve in the same way. You may get better at writing more details prompts or figuring out ways of getting what you want back from an LLM, but will you really understand the solutions that it comes up with? There may come a time when this isn't the case but, from what I've experienced, the more knowledge about a topic you have, the more useful AI is. This is true because AI "hallucinates", it's designed to predict the next best token in a sequence and feign intelligence, but this often results in subtly misleading information presented as facts. The more experience you have in a topic, the more likely you are to be able to see the issues and correct them. This experience is something you gain by dong, not by prompting.

As software engineers, we should be careful about automating the things that help us understand the problems we're trying to solve. AI can be useful in some circumstances, but to delegate the task of writing code to focus on "higher level" tasks might take us too far away from real understanding. Writing code and improving your craft is worth taking time, you can take shortcuts to a destination and miss the benefit the journey gives.