If You Use AI to Code, Use It to Learn Too

I've been building web sites for over 20 years. I remember the first reference book I got, *HTML for Dummies*. I remember getting the *HTML, XHTML, and CSS for Dummies* book the next year. I remember iFrames actually being considered good practice to not re-load image content you wanted persisted across pages (headers, nav, footers, etc). The standard screen resolution you developed for was 800x600.

If you were fancy you had a 1024x768 monitor!

Now I watch content creators on various social media platforms argue about vibe coding and the lost art of development vs velocity and getting things done.

And after 20 years, I get it.

20 years of experience with different languages, situations, products, industries, people, needs. 20 years of following the change as the internet adapted to people and people adapted to the internet. 20 years of honing my skills to craftsmanship levels.

And knowing that there are vibe coding solutions which are introducing long term maintainability issues, stability issues, security issues, that they don't even realize. Because they don't have the experience and knowledge to see it.

We can't go back. And I'm not arguing we should. The real question is — how do we move forward? As an industry of professionals, how do we move past the vibe code vs hand code debate and figure out how to use AI for its strengths, and use humans for theirs?

I think the first step, the right now moment for our industry, is for engineers to start using AI to learn MORE than they use it to write code.

In the past, if you went to HackerRank, CodeWars or Project Euler — they were sites with challenges that you either had to solve yourself, or try to find a solution online. And once you did…where was the feedback? I know that communities existed around them, but only a portion of engineers engaged in the community. Most wrote their code, happy to have the practice, and moved on. But at least they learned something.

Now, you can go to those sites and ask Claude or ChatGPT to just solve them and write the code for you. The LLMs probably have it saved somewhere in their memory from their training, and aren't even trying to work out the solution. It's a memory dump without any value, a fancy copy/paste.

And what do you learn from this? Less than before. You don't learn:

- How to spot performance issues by having your code time out.
- How to anticipate large data sets causing overflows, because you never experience it.
- What edge cases to consider in your code because the LLM copy/pasted a known solution that works for all of the test cases on the site.

You just hope you write a good prompt, hope you get a good result, and move on.

Instead, what if as an engineer, you take the time to solve it, and THEN talk to the AI? Feed it your solution. Ask it how it would rate you if it were a recruiter/interviewer. Ask it where you could have found better performance. Ask it if there are other known/common solutions. Be curious, and have a conversation with the AI. Verify what it says — do your own searches, read the docs, find the answers in the forums.

I propose that if you spend the time and energy to use AI as a tool for your own learning, you will grow faster, know more, and be able to work alongside LLMs more effectively. You will start to learn what real solutions look like. You will recognize when the AI starts to hallucinate. You will know when it's just dumping example solution code from a tutorial site modified to match your prompt and not real quality work that can stand up to production use.

So what does that look like, exactly? How can you effectively learn from AI when you can't necessarily trust that same AI to write the best code in the first place?

Set the model, extend the thinking: This is the perfect time to pick the best model and the extended thinking. You don't want it answering from memory. You want the LLM to actively plan its response actions, to conduct internet searches for information, and to validate its own responses. You can't have it run at the highest token eating levels all the time, but these types of requests are the perfect opportunity to get high value.

Engineer your prompt: Don't ask for a simple 'Examine my code'. Explain the context of what you are trying to accomplish. Tell the AI you don't want it to just dump all of the answers. Tell it that if you missed anything, you want help identifying it in the first place, and to ask questions to lead you there. Being able to think through the steps yourself is more valuable than rote memorization of a solution.

Follow up: The best result is that the LLM leads you somewhere new. You learn about a new primitive, or new method, or new algorithm. Think about where else what you learned might be useful, or what its limitations are — then ask the AI about those very thoughts. Are you on the right track? Are there other things you didn't think of? Keep going until you feel like you grasp those edge cases.

Ask for expert advice: Ask the LLM to verify its answers with industry experts, articles, research — and to provide you with links and sources. Take the time to validate the sources and read them yourself. You might discover something else new you didn't expect.

Make it a habit: Whenever you find something new — whether it's learning about a new library, or an old pitfall in the language that only the old coders of yore seem to remember — ask more questions, dig deeper, have conversations with the LLM. The AI won't judge you, you can feel free to ask whatever questions you need. And the more you learn, the better your questions will get.

We're all going to need to learn to manage Agentic flocks of LLMs in the future. And the engineers that understand how to parse if the flock is doing the right thing, the engineers that understand how to communicate in a way that the flock completes the task as intended — those are the ones that will have the most value.

If you're going to use AI to vibe code — don't forget to use it to learn as well.