My AI Adoption Journey

mitchellh.com/writing/my-ai-adoption-journey

February 5, 2026

▼ Table of Contents

My experience adopting any meaningful tool is that I've necessarily gone through three phases: (1) a period of inefficiency (2) a period of adequacy, then finally (3) a period of workflow and life-altering discovery.

In most cases, I have to force myself through phase 1 and 2 because I usually have a workflow I'm already happy and comfortable with. Adopting a tool feels like work, and I *do not* want to put in the effort, but I usually do in an effort to be a well-rounded person of my craft.

This is my journey of how I found value in AI tooling and what I'm trying next with it. In an ocean of overly dramatic, hyped takes, I hope this represents a more nuanced, measured approach to my views on AI and how they've changed over time.

> This blog post was fully written by hand, in my own words. I hate that I have to say that but especially given the subject matter, I want to be explicit about it.

Step 1: Drop the Chatbot

Immediately cease trying to perform meaningful work via a chatbot (e.g. ChatGPT, Gemini on the web, etc.). Chatbots have real value and are a daily part of my AI workflow, but their utility in coding is highly limited because you're mostly hoping they come up with the right results based on their prior training, and correcting them involves a human (you) to tell them they're wrong repeatedly. It is inefficient.

I think everyone's first experience with AI is a chat interface. And I think everyone's first experience trying to code with AI has been asking a chat interface to write code.

While I was still a heavy AI skeptic, my first "oh wow" moment was pasting a screenshot of Zed's command palette into Gemini, asking it to reproduce it with SwiftUI, and being truly flabbergasted that it did it *very well*. The command palette that ships for macOS in Ghostty today is only very lightly modified from what Gemini produced for me in seconds.

But when I tried to reproduce that behavior for other tasks, I was left disappointed. In the context of brownfield projects, I found the chat interface produced poor results very often, and I found myself very frustrated copying and pasting code and command output to and from the interface. It was very obviously far less efficient than me doing the work myself.

To find value, you *must* use an **agent**. An agent is the industry-adopted term for an LLM that can chat and invoke external behavior in a loop At a bare minimum, the agent must have the ability to: read files, execute programs, and make HTTP requests.

Step 2: Reproduce Your Own Work

The next phase on my journey I tried Claude Code. I'll cut to the chase: I initially wasn't impressed. I just wasn't getting good results out of my sessions. I felt I had to touch up everything it produced and this process was taking more time than if I had just done it myself. I read blog posts, watched videos, but just wasn't that impressed.

Instead of giving up, I **forced myself to reproduce all my manual commits with agentic ones.** I literally did the work twice. I'd do the work manually, and then I'd fight an agent to produce identical results in terms of quality and function (without it being able to see my manual solution, of course).

This was *excruciating*, because it got in the way of simply getting things done. But I've been around the block with non-AI tools enough to know that friction is natural, and I can't come to a firm, defensible conclusion without exhausting my efforts.

But, expertise formed. I quickly discovered for myself from first principles what others were already saying, but discovering it myself resulted in a stronger fundamental understanding.

1. Break down sessions into separate clear, actionable tasks. Don't try to "draw the owl" in one mega session.
2. For vague requests, split the work into separate planning vs. execution sessions.
3. If you give an agent a way to verify its work, it more often than not fixes its own mistakes and prevents regressions.

More generally, I also found the edges of what agents -- at the time -- were good at, what they weren't good at, and for the tasks they were good at how to achieve the results I wanted.

All of this led to significant efficiency gains, to the point where I was starting to naturally use agents in a way that I felt was no slower than doing it myself (but I still didn't feel it was any faster, since I was mostly babysitting an agent).

The negative space here is worth reiterating: part of the efficiency gains here were understanding when *not* to reach for an agent. Using an agent for something it'll likely fail at is obviously a big waste of time and having the knowledge to avoid that completely leads to time savings.

At this stage, I was finding adequate value with agents that I was happy to use them in my workflow, but still didn't feel like I was seeing any net efficiency gains. I didn't care though, I was content at this point with AI as a tool.

Step 3: End-of-Day Agents

To try to find some efficiency, I next started up a new pattern: **block out the last 30 minutes of every day to kick off one or more agents.** My hypothesis was that *perhaps* I could gain some efficiency if the agent can make some *positive progress* in the times I can't work anyways. Basically: instead of trying to do more in the time I have, try to do more in the time I don't have.

Similar to the previous task, I at first found this both unsuccessful and annoying. But, I once again quickly found different categories of work that were really helpful:

- **Deep research sessions** where I'd ask agents to survey some field, such as finding all libraries in a specific language with a specific license type and producing multi-page summaries for each on their pros, cons, development activity, social sentiment, etc.
- **Parallel agents attempting different vague ideas I had but didn't have time to get started on.** I didn't expect them to produce something I'd ever ship here, but perhaps could illuminate some unknown unknowns when I got to the task the next day.
- **Issue and PR triage/review.** Agents are good at using `gh` (GitHub CLI), so I manually scripted a quick way to spin up a bunch in parallel to triage issues. I would NOT allow agents to respond, I just wanted reports the next day to try to guide me towards high value or low effort tasks.

To be clear, I did not go as far as others went to have agents running in loops all night. In most cases, agents completed their tasks in less than half an hour. But, the latter part of the working day, I'm usually tired and coming out of flow and find myself too personally inefficient, so shifting my effort to spinning up these agents I found gave me a "warm start" the next morning that got me working more quickly than I would've otherwise.

I was happy, and I was starting to feel like I was doing more than I was doing prior to AI, if only slightly.

Step 4: Outsource the Slam Dunks

By this point, I was getting very confident about what tasks my AI was and wasn't great at. I had really high confidence with certain tasks that the AI would achieve a mostly-correct solution. So the next step on my journey was: **let agents do all of that work while I worked on other tasks.**

More specifically, I would start each day by taking the results of my prior night's triage agents, filter them manually to find the issues that an agent will almost certainly solve well, and then keep them going in the background (one at a time, not in parallel).

Meanwhile, **I'd work on something else.** I wasn't going to social media (any more than usual without AI), I wasn't watching videos, etc. I was in my own, normal, pre-AI deep thinking mode working on something I wanted to work on or had to work on.

Very important at this stage: turn off agent desktop notifications. Context switching is very expensive. In order to remain efficient, I found that it was my job as a human to be in control of when I interrupt the agent, not the other way around. Don't let the agent notify you. During natural breaks in your work, tab over and check on it, then carry on.

Importantly, I think the "work on something else" helps counteract the highly publicized Anthropic skill formation paper. Well, you're trading off: not forming skills for the tasks you're delegating to the agent while continuing to form skills naturally in the tasks you continue to work on manually.

At this point I was firmly in the "no way I can go back" territory. I felt more efficient, but even if I wasn't, the thing I liked the most was that I could now focus my coding and thinking on tasks I really loved while still adequately completing the tasks I didn't.

Step 5: Engineer the Harness

At risk of stating the obvious: agents are much more efficient when they produce the right result the first time, or at worst produce a result that requires minimal touch-ups. The most sure-fire way to achieve this is to give the agent fast, high quality tools to automatically tell it when it is wrong.

I don't know if there is a broad industry-accepted term for this yet, but I've grown to calling this "harness engineering." It is the idea that anytime you find an agent makes a mistake, you take the time to engineer a solution such that the agent never makes that mistake again. I don't need to invent any new terms here; if another one exists, I'll jump on the bandwagon.

This comes in two forms:

1. **Better implicit prompting (AGENTS.md).** For simple things, like the agent repeatedly running the wrong commands or finding the wrong APIs, update the `AGENTS.md` (or equivalent). Here is an example from Ghostty. Each line in that file is based on a bad agent behavior, and it almost completely resolved them all.

2. **Actual, programmed tools.** For example, scripts to take screenshots, run filtered tests, etc etc. This is usually paired with an AGENTS.md change to let it know about this existing.

This is where I'm at today. I'm making an earnest effort whenever I see an agent do a Bad Thing to prevent it from ever doing that bad thing again. Or, conversely, I'm making an earnest effort for agents to be able to verify they're doing a Good Thing.

Step 6: Always Have an Agent Running

Simultaneous to step 5, I'm also operating under the goal of **having an agent running at all times.** If an agent isn't running, I ask myself "is there something an agent could be doing for me right now?"

I particularly like to combine this with slower, more thoughtful models like Amp's deep mode (which is basically just GPT-5.2-Codex) which can take upwards of 30+ minutes to make small changes. The flip side of that is that it does tend to produce very good results.

I'm not [yet?] running multiple agents, and currently don't really want to. I find having the one agent running is a good balance for me right now between being able to do deep, manual work I find enjoyable, and babysitting my kind of stupid and yet mysteriously productive robot friend.

The "have an agent running at all times" goal is still just a goal. I'd say right now I'm maybe effective at having a background agent running 10 to 20% of a normal working day. But, I'm actively working to improve that.

> **I don't want to run agents for the sake of running agents.** I only want to run them when there is a task I think would be truly helpful to me. Part of the challenge of this goal is improving my own workflows and tools so that I can have a constant stream of high quality work to do that I can delegate. Which, even without AI, is important!

Today

And that's where I'm at today.

Through this journey, I've personally reached a point where I'm having success with modern AI tooling and I believe I'm approaching it with the proper measured view that is grounded in reality. I really don't care one way or the other if AI is here to stay, I'm a software craftsman that just wants to build stuff for the love of the game.

The whole landscape is moving so rapidly that I'm sure I'll look back at this post very quickly and laugh at my naivete. But, as they say, if you can't be embarassed about your past self, you're probably not growing. I just hope I'll grow in the right direction!

I have no skin in the game here, and there are of course other reasons behind utility to avoid using AI. I fully respect anyone's individual decisions regarding it. I'm not here to convince you! For those interested, I just wanted to share my personal approach to navigating these new tools and give a glimpse about how I approach new tools *in general*, regardless of AI.

1. Modern coding models like Opus and Codex are specifically trained to bias towards using tools compared to conversational models. ↩
2. Due to the rapid pace of innovation in models, I have to constantly revisit my priors on this one. ↩
3. The skill formation issues particularly in juniors without a strong grasp of fundamentals deeply worries me, however. ↩
4. I don't work for, invest in, or advise any AI companies. ↩

February 5, 2026