

The Detroit mistake

Varun Choraria · 2026-10-25 · <https://www.varunchoraria.com/the-detroit-mistake/>

I remember chasing technical writers across companies, especially as a new joiner, to get the product documentation updated. When I didn't have the means to buy courses, I learned by going down the rabbit holes of documentation companies put out. I still enjoy that. To me, it's the only positive indicator of sustainable product development.

With AI, the days of taking documentation and version control for granted are gone. One reason this matters again is that the product is no longer built only for the user. It's also built for agents, because products now have agent-driven workflows.

The commit is no longer the whole record

Take version control. Commits mostly logged the change in the codebase after every pull request. Testing sat apart from that, mostly internal logs or slide decks. What's changing, because AI is writing and reviewing most of the code, is that we're holding it accountable for more than the diff. We're asking whether it works as intended, and whether that lines up with user behavior and goals.

On my own sites I keep a database of logs, and something like a dynamic memory. That memory is what gives the agent the human habit of looking back at what it already did. Before a setup like that existed, the workaround was a `soul.md` file and a skill. I triggered it by hand at first, or let it run on a weekly schedule, so it could review past decisions, log silent bugs and workflow redundancies, and suggest fixes for me to review. It also wrote the later decisions back into its memory, so it would not repeat the same mistakes. I'll admit it wasn't perfect.

That's automation. AI is the means to an end.

The Detroit mistake

Half the internet is talking about building more complex systems, or more efficient ones. The other half is still trying to find the business return on a ChatGPT project they set up 2 years ago.

I found an article that names the thing: slop. I've read a lot about Toyota and Ford's shop-floor automations, built to produce more cars, faster. More cars meant more sales. That only holds if the demand was already there to justify making more cars.

Apparently GM's car and truck market fell by nearly 10% in 10 years, despite all that automation.

I lived a smaller version of this. At a previous organization, a team of three spent upwards of \$20K a month on AI tools. What followed was the usual bell curve. At first, a ton of hype. It was as though we'd discovered fire, and were now in charge of figuring out the use cases. After a month or two, the job was keeping that fire burning, because a lot of chopped wood had been bought at a premium, and we had to keep finding ways to make the fire useful. In that quest, we forgot the customer and the prospect.

You can keep the circus running and still be building the wrong thing, just a lot faster.

Every company serves a unique customer in a unique market. Don't build software to catch up with a competitor, or because it's what you think customers want. The clear return I've actually seen is automating low-value tasks so you can go to market faster and serve the customers better.

Spec is the new source code

[OpenAI's account of forward deployed engineering](#) is about closing the gap between what a customer expects and what gets delivered. Pick their hardest problems. Build a repeatable solution, so the next time starts from a higher point. That last phrase is the one I keep. I'm starting to believe every quarter is a new year. Things no longer end when finance says they do.

The other admission, as I read it, is that frontier models will keep showing up, and that is not a guarantee they fulfill a real-world outcome. Like a coin flip, the odds of one-shotting anything with Fable 5.1 are still 50:50. The number of tokens is a false positive. What matters is cost per task.

At work I use Notion as my agent database. IT didn't block it. I run a set of Claude routines that take the admin work:

1. Map the political landmines and help me draft responses. The org is a giant matrix, with more dotted lines than I can keep in my head.
2. Map the key departmental contacts and build the rolodex. Read the email trails, and work out who to reach first and whom to cc.
3. Every morning, ingest the last 2 weeks of Teams and Outlook messages, log them, and build the day's worklist as tickets. Tag each ticket and update it with the latest action items and an audit trail, so nothing gets dropped.
4. Write a weekly email to my reporting manager: what got done, what to watch, and what's next.

There's more than that. I have a \$500 budget on the enterprise plan each month. Most of the execution, a lot of API calls every day and still inside Notion's free-tier limits, costs me \$70 a month. That's a fraction of a virtual assistant, and I'm not drowning in sticky notes or fishing through Outlook. The Outlook UX is horrendous. Most of the task execution runs on Sonnet 5.5 extra.

When a more efficient frontier model shows up, I can pull routines off and put new ones on, and compare them to the dollar cost. The next time I ask for a limit extension, or have to justify the spend, I have crude numbers. The next time I sit down to improve the workflows, the spec starts from those points: tasks on one axis, model on the other, a trend line over the busiest days, and what actually got done. Volume of work, setup, config.

The other thing OpenAI has been quiet about is that this behaves a lot like a services company. I'm pretty sure the forward deployed engineers embedded with a specific client are charging by the hour, the way a managed-services or transformation business does. In one case they embedded with a semiconductor client, studied semiconductors on the side, and built evals and workflows across that client's use cases.

The model, the tool, is a delivery workbench.

Underneath it, the spec is the old one. Problem, constraints, architecture, workflow, value. That document is the precursor for whatever comes next.