Vibin Labs

The Fintagonist

Opus 4.8, Fable 5 and GPT Lie With Impunity - Software Manufacturing Guidelines

If it's Saturday (or Tuesday), it must be the Fintagonist. Fin, Pro, An, The Fintagonist, a contrarian's view on software manufacturing with artificial intelligence in the financial services and fintech space. Bringing unrequested insights to the world…

Austin, TX, July 11, 2026

This week in the news… It's 7-11. Come on, that's always cool.

It's been a quiet week for the AI doomers. Jobs seem to not be in danger this week. Some companies are hiring people back??? And token-maxxing is officially passe. Now we have shifted to token-economics as companies find cheaper ways to waste tokens on wild unruly agents.

So, we shall take a technical approach to this week's posting. This is for the folks manufacturing software right now. Today we will discuss and debate the guidelines that help produce quality manufactured software. Boring for some, more critical for those actually producing software.

Here they are, the 12 practical canons of software manufacturing! And for those who have developed software using the traditional hand-coding process, these all sound mysteriously familiar…

1. Write Requirements Not Prompts

Write a requirements document. Write up your detailed spec and use cases. Build a prototype. Don't just go to the LLM prompt and type in some high-level generic statement of what you want and expect the LLM to create it successfully. If you could give the LLM a hundred pages of description, do it. The more you document, the better the outcome. And specifically create a requirements inventory as you will use that throughout the process.

2. Always Have LLM Write A Plan

Always make sure the LLM session creates a step-by-step plan for any substantial amount of work. Make sure you see it on the screen and make sure it exists in a document the LLM can refer to. Make sure to cross items off this plan. And make sure you monitor the progress. Do not just rely on the LLM to tell you where you are. If requirements change within the process, make sure the plan is updated with any new

necessary steps. The LLM will do all the planning work and the person verifies the results.

3. Define Architecture Upfront

Define the exact software architecture upfront. This is not choosing a specific technology as much as choosing at a high level what you want. Is it a web app? Is it hosted? Is it running at a business or used by individual consumers? These upfront decisions will dictate a specific path technologically once the LLM gets engaged to figure out how to manufacture the software. These decisions become part of the overall requirements.

4. Audit First Process

The 400+ audit standards will be fed to the LLM before the project starts. This audit-first approach lets the LLM know what is coming and they will plan for it. The LLM will constantly be referring to the audit standards. And the adherence to the standards increases dramatically when the LLM has visibility to them upfront. You still need to audit 2-3 times during the manufacturing run and at the end, but overall, the process is improved with the audit-first approach.

5. Verify After 1-2 Steps Never Longer

Verify LLM output after 1-2 steps. Do not go longer than 1-2 steps without checking the work. Verification can be brief and it is necessary. An LLM going longer than 1-2 steps will drift and will hallucinate and will basically lower the quality of the work. It is painless and only mildly annoying to have to do this. And if I hear the LLM catchphrase “it’s on me,” I think I will get physically sick.

6. Giga File Structure

Tell the LLM to put the entire product in a single file. They won’t. It’s impossible to have a single file, but you will get many fewer files. LLMs like one big file. It keeps them in context, and they don’t need to go hunting for different files. The giga file is well organized and designed for the LLM. A hand coding human would not like this file but it’s ok, they never look at it… Fewer files also guarantee the LLM will not attempt to rewrite the file (see below), so the structure is self-policing. Long live giga-files…

7. Common Services Defined Upfront

Make sure the LLM creates a set of common services used throughout the app. Enough said on this one as, frankly my dear, it’s way beyond my technical know-how. But it works and it is just good tech architecture.

8. Requirements Audit at Key Stages

Check the work against the requirements inventory and do it often. This one is hugely important. See, the LLM is very optimistic. It thinks it's done and it's not. A simple check against the requirements inventory yields results every time. And you have to verify this often: verify the plan, verify the phases and verify the end result. Act often and trust not the LLM, for they lie with impunity if not bad intent. The LLM is designed for summaries and brevity. It doesn't really like finishing its work. So, check the work, make sure the requirements have been manufactured in the code. When asked, the LLM is very honest "Hey Robb, it's on me." Barf!!!

9. Never Rewrite Code

Make sure the LLM edits the code, not rewrites it when you are making iterative changes. The LLM will rewrite existing code rather than edit it. It's a bad habit as then it drops functionality, almost every freaking time. So always surgically edit, never rewrite. The giga-file structure helps here as when the file is large, the LLM will self-realize that a rewrite is not practical, so you force it to edit and iterate… Tricky, tricky, to use the process and structure to manage the LLM in a less obvious manner!!!

10. Never Use Sub-Agents

Just don't do it. It wastes huge amounts of time and tokens. Get the primary LLM session to do the research and the actual work. The primary session invariably replicates the work of the sub-agent. Example: when the LLM primary session takes advice from a sub-agent on the code, it still needs to then do its own investigation before it changes anything. It's annoying to watch a sub-agent summarize something and then the primary LLM go in and find the same thing. Not as annoying as the rampant token wastage but pretty darn annoying.

11. Force LLM to Verify Not Use Inference

This is a subtle one. The LLM has 2 modes. In one mode it actually reads the info, the document, maybe does a web search. It then gives you facts based on what it learned. It verifies the sources. The other mode is super annoying, and it is this mode that creates the well-intentioned and yet lying LLM. When the LLM uses inference, they make decisions based on the chat history and the data in their model, both of which can be suspect sources. The LLM does not really make things up. They just make things up based on those 2 data sources. And they don't lie with bad intent. They just state facts with such certainty when they have not checked their work. It's on me, Robb…

12. Dueling LLMs - Break Bug Cycles With 2nd LLM

And lastly, the dueling LLMs… Pit the LLMs against each other, especially if you've hit a brick wall fixing a bug. A second opinion by a different LLM finds issues really quickly. This same process is used for the audit-first process. One LLM audits, a second LLM audits the audit. An LLM can act just like a person who is too close to a problem to solve

it, and at times the LLM is too close as well. So, break the cycle. Come up for air. And use the services of a dueling LLM to keep the process honest.

Enough said on this topic. Next week the Fintagonist will be back with interesting general-purpose content to be ignored by the masses!