Claude plans, Codex builds — the plan→review→build setup

You commented GRILL. Here's the whole thing — the real repo, the exact install, and the four steps that take a vague idea to verified code with two models checking each other instead of you babysitting one.

The thing that actually breaks

You type a rough idea into your AI coder and say "build it." It builds *something*. Fast. Confident. And then you spend the next hour finding out it guessed wrong about the one thing that mattered — the auth flow, the edge case, the assumption you never spelled out because you didn't know you had to.

That's vibe-coding. One model, one shot, no plan, and you as the only reviewer. It feels quick until you count the time you lose cleaning up after it.

The fix isn't a better prompt. It's a second opinion *before* a single file gets written — and a plan good enough to be worth reviewing in the first place.

What it looks like when it works

One idea goes in. What comes out the other side is one log that reads like this:

```
grilled   → PLAN.md locked (Goal / Approach / Key decisions /
Risks)
reviewed  → Codex read the whole repo, read-only, and pushed
back
verdict   → REVISE ×2, then VERDICT: APPROVED
built     → Codex wrote the code, Claude read the diff like a PR
verified  → proof test passed
```

No "build it and hope." The plan got interrogated, a second model hunted it for holes, the holes got fixed, and only *then* did anyone touch the code. You approved the plan once and read one diff at the end. That's the whole job.

The repo (this is the real tool)

`grill-me-codex` — github.com/chaseai-yt/grill-me-codex — **615★, MIT, open source.**

It's a Claude Code skill. Installing a skill is one move: **you drop its folder into `~/.claude/skills/`.**

```
git clone https://github.com/chaseai-yt/grill-me-codex
cp -r grill-me-codex/grill-me-codex ~/.claude/skills/
```

Restart Claude Code (or open a fresh session) and the skill is live. On Windows that skills folder is `%USERPROFILE%\.claude\skills\`. Drop the folder in, it's there. Delete it, it's gone. No config file.

You'll also need Codex reachable from your machine — that's the second model that does the reviewing and, later, the building.

Step 1 — GRILL: Claude drags the plan out of your head

You don't hand it a spec. You hand it a rough idea, and Claude interviews you — **one question at a time**, and every question comes with a recommended answer so you're never staring at a blank prompt. You either take the recommendation or correct it. Next question.

It keeps going until nothing important is still fuzzy, then it writes everything into `PLAN.md` with four parts:

- **Goal** — what you're actually trying to make true.
- **Approach** — how it's going to get there.
- **Key decisions** — the forks it hit and which way it went, on the record.
- **Risks · Out of scope** — what could bite, and what you're deliberately *not* doing.

That last part is the quiet hero. Half of bad builds come from the model inventing scope you never asked for. Writing "out of scope" down kills that before it starts.

By the end of step 1 you have a plan a stranger could read and understand. That's the point — because a stranger is about to.

Step 2 — REVIEW: the plan goes to Codex, read-only

The locked `PLAN.md` gets handed to Codex in a **read-only sandbox.** This matters: Codex can read your *entire* repo — every file, how it all connects

— but it **cannot write a single file.** It can't "helpfully" start coding. Its only job is to find what's wrong with the plan.

And it goes looking for the stuff that actually hurts:

- security holes
- race conditions
- missing edge cases
- wrong assumptions baked into the approach

Every reply it gives ends with exactly one line, no hedging:

```
VERDICT: APPROVED
```

or

```
VERDICT: REVISE
```

So there's never a "well, sort of, maybe" — you get a clean signal every round.

Step 3 — VERDICT: fix, resume, repeat (Claude stays in charge)

If the verdict is **REVISE**, Claude takes Codex's critique, fixes the plan, and **resumes the same Codex session** — not a new one. That's deliberate: Codex remembers every point it already raised, so it's checking whether you *actually* fixed it, not re-reading cold and forgetting what it said last round.

Two rules keep this from going sideways:

- **Claude is the final arbiter.** Codex advises; it doesn't get a veto. If Claude thinks a critique is wrong, it says why and moves on. You're not handing the wheel to the reviewer.
- **Hard cap: 5 rounds.** If a plan can't get to APPROVED in five passes, that's the system telling you the idea itself needs a rethink — not more polishing. No infinite loops.

You watch this happen and sign off. The plan that comes out the far side has been through a real adversarial read, not a rubber stamp.

Step 4 — FLIP: roles reverse, code gets written and checked

Here's the move that makes the whole thing click. Once you sign off on the plan, **the roles flip.**

- **Codex writes the code** — now with full access, building against the plan you both already agreed on.
- **Claude reads the entire diff like a PR** — the way a senior dev reviews a junior's pull request. Every change, against the plan.
- Then Claude **runs the proof test** — the actual check that the thing does what step 1 said it would.

The model that critiqued the plan is now the builder; the model that wrote the plan is now the reviewer. Each one checks the other's strongest work. You get one diff to glance at and a proof test that either passes or doesn't.

One idea in. Grilled → reviewed → built → verified out. One log for the whole trip.

A bonus you'll feel in your bill

Offloading the actual building to Codex means Claude isn't burning its context writing hundreds of lines of code — it's reading and judging, which is cheap. **You save your Claude tokens for the thinking**, and hand the heavy typing to the other model. Two brains, and the expensive one stays free for the part that needs it.

Why plan-first beats vibe-coding (the honest version)

Vibe-coding isn't bad because the code is bad. It's bad because there's **no checkpoint before the mistake becomes files on disk.** One model, guessing at scope, with you as the only reviewer *after* it's already built the wrong thing.

Plan-first flips the order. The cheapest place to catch a wrong assumption is in a plan, before any code exists — a bad line in `PLAN.md` costs ten seconds to fix; the same bad assumption caught after the build costs you an afternoon. Adding a second model that *only* attacks the plan means the mistakes get caught while they're still cheap. The build at the end is almost boring, because everything hard already got argued out.

You're not slower for planning first. You're slower for *not*.

The bigger picture

This is a small piece of how I actually build things at Axionox — real production systems for clients, not no-code duct tape. The whole game is getting AI to check itself so a human doesn't have to catch everything by hand. `grill-me-codex` is that idea shrunk down to one skill you can run tonight: two models, one planning, one attacking, then they swap and build.

Once you've felt it on one small feature, you start wanting every serious build to go through a plan→review gate. That instinct — never let the model build straight from a vibe — is the thing worth keeping, whatever tools you end up using.

Your next step

Don't reorganize your whole workflow tonight. Pick one small thing you were about to "just build" — a script, an endpoint, a fix — and run it through the loop instead. Clone the repo, drop the folder in `~/.claude/skills/`, restart, and start with the grill.

```
git clone https://github.com/chaseai-yt/grill-me-codex
cp -r grill-me-codex/grill-me-codex ~/.claude/skills/
# restart Claude Code, then point the skill at your idea and let
it grill you
```

Do it once on something small. Watch Codex catch a hole you'd have shipped. That's the moment it clicks — and after that you won't want to build the other way.

Built by Saad Bhatti — Axionox. I build production AI systems for a living (axionox.com). Follow for more like this.