

Anthropic’s Scion lessons: tell Claude no, and watch adversarial review for overengineering

Coding Agents Alpha Tracker

2026-10-16

Anthropic’s Scion lessons: tell Claude no, and watch adversarial review for overengineering

By Coding Agents Alpha Tracker • October 16, 2026

What Anthropic learned building its git replacement with Claude, the token cost of Claude Code’s write/verify/fix loops, ways to stop agents waiting on tool calls, and Armin Ronacher’s Pi workflow. Also covers T3 Code leaving alpha, wider Argon access, Muse Wonder in Cursor and Claude Code Review going GA.

What Anthropic learned building Scion with Claude

The last brief covered Anthropic moving its monorepo off git. Theo’s walk-through of Anthropic’s write-up now adds lessons on how the engineers worked with Claude. Every engineer on the project now rates Claude as better at writing code than they are. Their value was in system-level understanding and design trade-offs, and very often their job was “to tell Claude no.” Agents sometimes broke core system constraints. For example, during migrations several systems tried to find when a change merged by reading the commit log linearly instead of asking the code review tool [1].

Two practices are worth adopting:

- **Watch adversarial review for scope creep.** Anthropic uses adversarial review heavily, but it can be short-sighted. In one case a review was overly defensive about trusting other internal services, and Claude added excessive validation and complexity. Engineers had to steer it away from hypothetical concerns. Theo adds that this is worse when the feedback comes from other model families. Reviewers still look at every change request, but they check the *intent* of the change rather than reading every line [1].

- **Test after merge instead of locking the merge queue.** Anthropic could not make a global repo lock on the merge queue work at agent scale. It invested in automated testing after merge and before deploy instead. Concurrent changes sometimes conflict, but Claude can usually fix them faster than humans can even spot them [1].

Two smaller points. When given an early snapshot of Mythos Preview and no further instructions, it rewrote a hand-written ingestion prototype to use 20x less memory and run 4x faster, while keeping its ordering and data-structure invariants [1]. Theo measured that almost 1,000 hours of T3 Code’s CI time went to waiting on git checkouts [1].

Verification loops cost a lot of tokens

Simon Willison has been running Claude Code in “Ultracode mode” [2]. He gave it one prompt, “Write a SQLite extension in Rust that implants SPARQL...”, and it was only finishing 12 hours later [3]. The catch is token use. The mode puts everything through a write/verify/fix cycle, and adding READMEs alone had used 3.4M tokens. His screenshot shows 22 agents in write, verify and fix phases. Each writer used about 140K–270K tokens per crate [4]. Expect thoroughness to cost you, and save fan-out workflows for tasks where verification pays for itself.

Agents spend much of their time waiting on tool calls

A post about Peter Steinberger’s Crabbox argues that agents now spend more time waiting on tests and builds than writing code. Its checklist [5]:

1. Measure which tool calls are slow.
2. Run only the relevant tests, not the whole suite.
3. Run independent checks in parallel.
4. Move CPU- or RAM-heavy work to other machines with Crabbox.
5. Use faster tools, such as Oxlint instead of ESLint. The authors’ own Swift linter was 20x faster.

Armin Ronacher: parallel checkouts, small cores and compiler checks

On Pi itself, **Ronacher** works conservatively. He runs several checkouts with parallel Pi instances to investigate issues and open tickets, and keeps Pi running on a machine that stays on when his laptop is closed. He says he hasn’t figured out subagents. He is also cutting back on keeping agents busy for its own sake, after finding that some of it was “a horrible waste of tokens” [6].

His view on rules files: Pi Durable is small enough that an agent can load all of it into context and understand it. A small, well-designed core needs little guidance, while a codebase with millions of lines does need concise rules or skill

files [6]. He also prefers compiler checks to tests, because agents “write a lot of shitty tests” and then run only a small subset of them. They also tend to run code before type-checking it [7].

Pi Durable is his redesign of the agent core. It saves state after every step and models each piece of work as a task, so it can tell an unfinished run from a completed one and resume after a shutdown. It also allows background work such as asynchronous compaction [7].

For his kids’ game, he embedded an in-browser agent in an interactive guide that runs the game’s real code. The agent acts as a tutor and knows what page the learner is looking at [8]. He also added **learn:** references in the code so the agent would update the docs as it edits, but he says it only does this sometimes [8].

Skills: delete what models now do on their own

- **Kent C. Dodds** deleted his orchestration skill because agents now manage orchestration well by themselves (PR) [9].
- **poteto’s pstack** is a frequently updated skill set for rigorous, autonomous engineering. Run `/setup-pstack` and pick a budget, put `/poteto-mode` in front of a prompt you want done rigorously, and use `/poteto-help` when unsure which skill applies. It is available for Cursor and Grok Bot, with source here [10].
- **Theo** asked OpenAI’s Tibo for skills that behave differently depending on the model, and that change their advice by mode, such as avoiding slow git commands in ultrafast mode [11].
- **Ben Tossell** uses Pi for everyday questions because it starts with clean context, Factory when he wants well-built code, and the BB app to run Codex, Claude, Pi and Factory from one interface [12].

Releases and availability

- **T3 Code** is out of alpha. Orchestrator V2 is on the stable channel as 0.1.0 Beta, and the 2.0.0 mobile apps are on the App Store and Play Store [13]. Jacob Wolf demoed it to beehiiv’s engineers with a build loop that moves across Claude, Cursor, OpenCode and the CodeRabbit CLI. He called its output the most production-ready AI code he has read [14].
- **Gemini 4 Argon** is rolling out in Antigravity to paid Google AI Pro subscribers. It has its own weekly-only quota, separate from the existing Gemini 3.8 Flash quota [15]. Given last brief’s hallucination findings, verify its claims.
- **Muse Wonder** is now in Cursor. It scores 46.1% on CursorBench at default effort, the highest of any Muse model, and costs \$4/M input and \$20/M output [16].
- **Claude Code Review** is GA for Team and Enterprise plans, with Quick, Standard and Deep levels. **Opus 5.5 fast mode** is GA with no waitlist,

running up to 2.5x faster at \$8/\$40 per 1M tokens [17].

- **Codex**: OpenAI’s Tibo says changes and extra compute should make “Selected model is at capacity” errors much rarer. When they do happen, sessions should eventually continue on their own instead of blocking [18].
- **Geoffrey Huntley** renamed Kittens to **Overalls**, an MIT-licensed, self-hosted AI developer platform he plans to release at AI Native Dev in New York around Nov 3 [19]. It now implements Amp threads and Orbs. Next he plans to tune its system prompt and tooling and have it build itself: “Goodbye Codex” [20].

Sources

1. Anthropic Rebuilt Git (yes really)
2. X post by @simonw
3. X post by @simonw
4. X post by @simonw
5. X post by @superlinear_fm
6. React Over Solid? with Armin Ronacher
7. React Over Solid? with Armin Ronacher
8. Teaching Kids Programming with Pi
9. X post by @kentcdodds
10. X post by @poteto
11. X post by @theo
12. Non-Technical Founder: Don’t Need to Know How To Code. You Need to Start - Ben Tossell
13. X post by @jullerino
14. X post by @JacobWolf
15. X post by @antigravity
16. X post by @cursor_ai
17. [AINews] Agents disprove Polysemanticity; greatly improve Interpretability & Alignment
18. X post by @thsottiaux
19. X post by @GeoffreyHuntley
20. X post by @GeoffreyHuntley