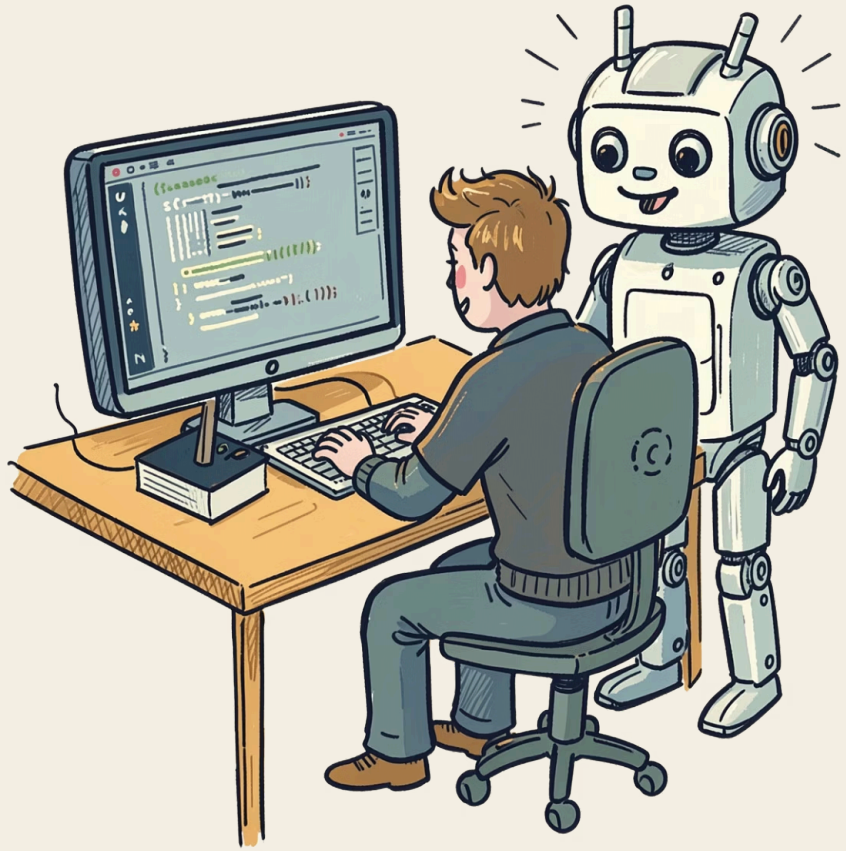


Building the AI-first Developer Stack

@avindrafernando

July 2025

Day 9 of a 12-day public experiment.

Who Am I?

Avindra Fernando

Principal Architect/Fractional CTO, Taprobane Consulting

@avindrafernando | taprobaneconsulting.tech

"Raise your hand if AI helps you write code."

"Keep them up... if AI refines your tickets before anyone plans"

"Keep them up... if AI writes a plan you approve before any code exists"

"Keep them up... if agents review the PR before a human does"

"Keep them up... if AI writes and heals your end-to-end tests"

We adopted AI at exactly one stage.

We called this "AI-assisted development" in 2024.

Chat tab

Ctrl+C

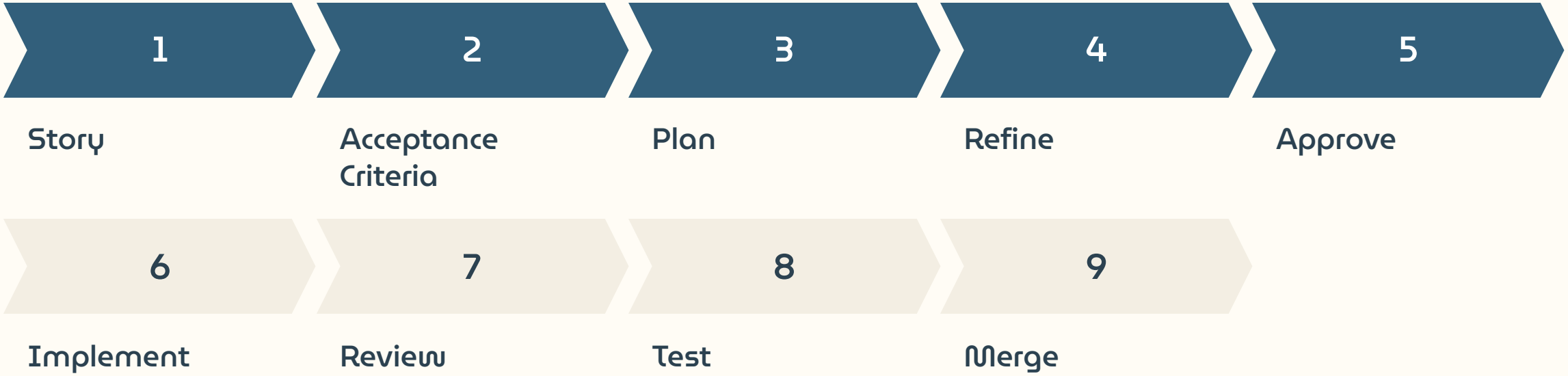
IDE

Ctrl+V

In 2026 the reusable intelligence lives in files.

```
repo/
 ├──── AGENTS.md
 ├──── .cursor/rules/
 ├──── .claude/skills/
 ├──── plans/
 │    └──── export-feature-plan.md
 └──── specs/
  └──── export-feature.md
```

Upstream


1
Story
2
Acceptance Criteria
3
Plan
4
Refine
5
Approve
6
Implement
7
Review
8
Test
9
Merge

The Stack

1. **Context**
   rules, docs, conventions
2. **Planning**
   refine, plan, approve
3. **Execution**
   agents, models, cloud
4. **Guardrails**
   permissions, sandboxes, hooks
5. **Validation**
   types, lint, tests
6. **Review**
   agents, bots, humans
7. **Reusable memory**
   skills, plans, playbooks

Context: the file the agent reads first.

```
# AGENTS.md

This project uses React with TypeScript. Testing is done using Playwright (E2E) and Vitest (unit).
It also uses TanStack Query for data fetching and MUI for UI components.

##  General Guidelines

- Use **TypeScript** across all files, where applicable.
- Stick to **function components** and **React hooks**.
- Keep components small and reusable.
- Use **TanStack Query (React Query)** for data fetching (found mostly in `src/hooks/react-query`).
- Wrap network calls with custom hooks (e.g., `useUser()` instead of calling `useQuery` directly in components).

## Testing Guidelines

### Unit Tests (Vitest)
- Use `.test.ts(x)` files colocated with source.
- Use `describe`, `it`, `expect` from `vitest`.
- Mock React Query hooks using `vi.mock(...)`.

### E2E Tests (Playwright)
- Store tests in `e2e/` folder.
- Prefer role-based, label-based, placeholder-based, text-based queries (in that order), over `page.locator` queries.
```

Best practices I've landed on.

- Commands verbatim, runnable as written
- Conventions with the why
- Boundaries: what never to touch
- Short beats complete: link out
- Update it every time the agent gets something wrong

Stage one: the ticket.

```
# TICKET: Customers should be able to export their history
Requested by: Sales
Priority: High
Acceptance criteria: (none)
```

The agent reads the ticket against your codebase.

Refined acceptance criteria
- Export formats: CSV and PDF
- Date-range filter, default 12 months
- Async generation for large accounts
- Email notification with download link

Open questions
- Include voided records?
- Retention period for generated files?

The plan is markdown. Committed next to the code.

 # Plan: Export feature

Approach
Async job via existing queue. Wrap the existing PDF service, do not modify it.

Files
- app/Jobs/GenerateExport.php (new)
- app/Services/ExportService.php (new)
- routes/api.php (add 2 endpoints)

Tests
- Unit: export service formats
- E2E: request export, receive link, download

Out of scope
- Scheduled/recurring exports

Iterate on the plan, not the diff.

Reject

"Don't touch the PDF service. Wrap it."

Question

"What happens when the queue is down?"

Regenerate

New plan in seconds.

How certain are you?

"Ask before you approve. Ask again before you merge."

"Approve the plan before you pay for the diff."

Let's do it live.

The Assembly Line

1	2	3	4	5
Story ✓	Acceptance Criteria ✓	Plan ✓	Refine ✓	Approve ✓

6	7	8	9
Implment ✨	Review	Test	Merge

My setup: three agents, three jobs.

Planner

plan mode, strongest model, full context

Implementer

executes the approved plan, faster model

Reviewer

fresh session, review pass before the PR

My Team

Planner Agent

Implementer Agent

Reviewer Agent

↑ ↑ ↑

Me

Ticket in. Draft PR out.

1

Assign the task

2

Cloud agent works on a branch in isolation

3

You review a draft PR

Remote control.

Autonomy without boundaries is not a workflow. It is a risk.

Read scope	Write scope
Commands & network	Secrets
Approval gates	Spend limits

Day 9. The code freeze was written in all caps.

The most popular skill in the community is three sentences.

```
---
name: grill-me
description: Interview the user relentlessly about a plan or design
  until reaching shared understanding, resolving each branch of the
  decision tree. Use when user wants to stress-test a plan, get grilled
  on their design, or mentions "grill me".
---
Interview me relentlessly about every aspect of this plan until we
reach a shared understanding. Walk down each branch of the design tree
resolving dependencies between decisions one by one.
```

grill-me, Matt Pocock, github.com/mattpocock/skills (MIT)

🎬 What getting grilled looks like.

The three skill categories I recommend starting with.

Conventions

how this repo works

Workflows

repeatable multi-step jobs

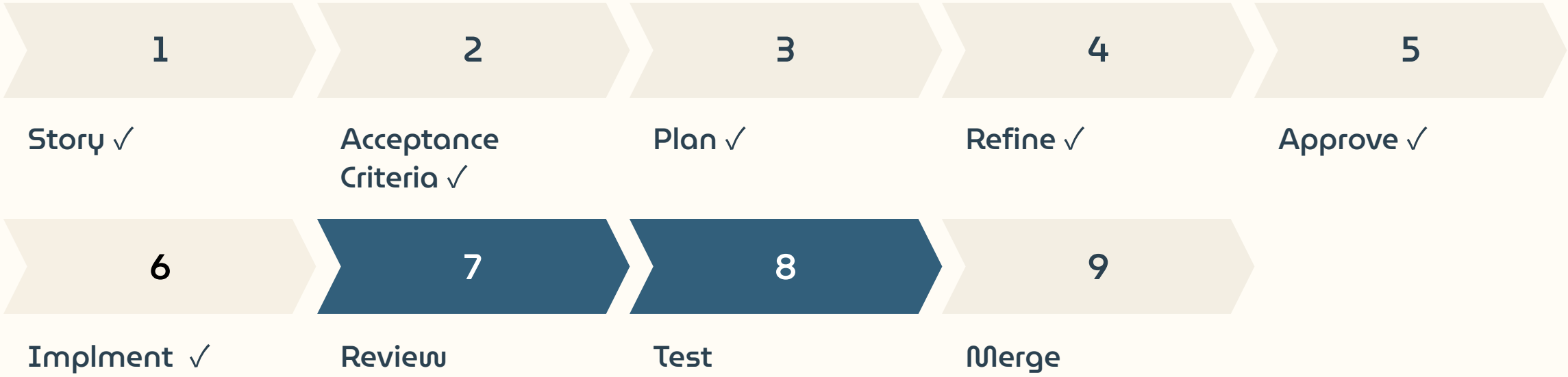
Process

commits, PR descriptions, checklists

We watched an agent "refuse" to code.

If a user requests a second export while the first is still generating, what happens? Cancel the first, queue the second, or reject it? And what does the user see in each case?

Trust the Line



Three gates before a human spends attention.

In the editor

review passes before the PR exists

On the PR

bots comment within minutes

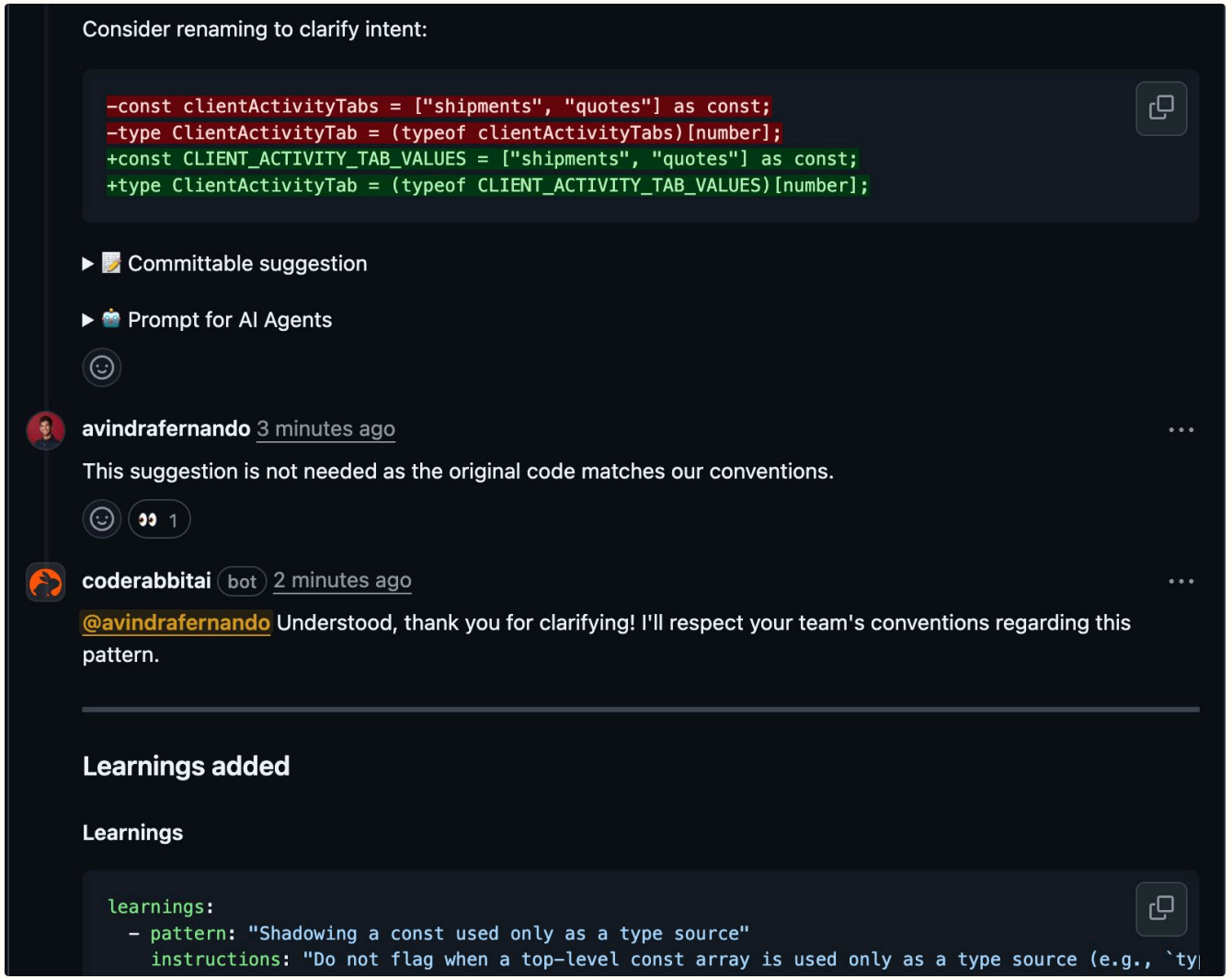
The human gate

intent, architecture, risk, accountability

Agents fail differently.

- Did it run the tests it claims?
- Files outside the approved scope?
- Weakened or deleted assertions?
- Errors hidden rather than fixed?
- Does the diff match the plan?
- Unnecessary dependencies?

What I do with bot feedback.



Tests can start in plain English now.

Then

```
await page.locator('#export-btn-2').click()
```

Now

"Request an export, receive the email, download the file."

From test-driven to spec-driven.

"TDD: the test drives the code."

"SDD: the spec drives the plan, the code, and the tests."

Planner. Generator. Healer.

🎭 Planner

explores your running app, writes a markdown test plan

🎭 Generator

turns scenarios into test code, checking the live app

🎭 Healer

investigates failures, proposes repairs

⚠️ A healed test can prove less than it did.

Readable by your PM. Traceable to the story.

From the AC: "Async generation for large accounts"

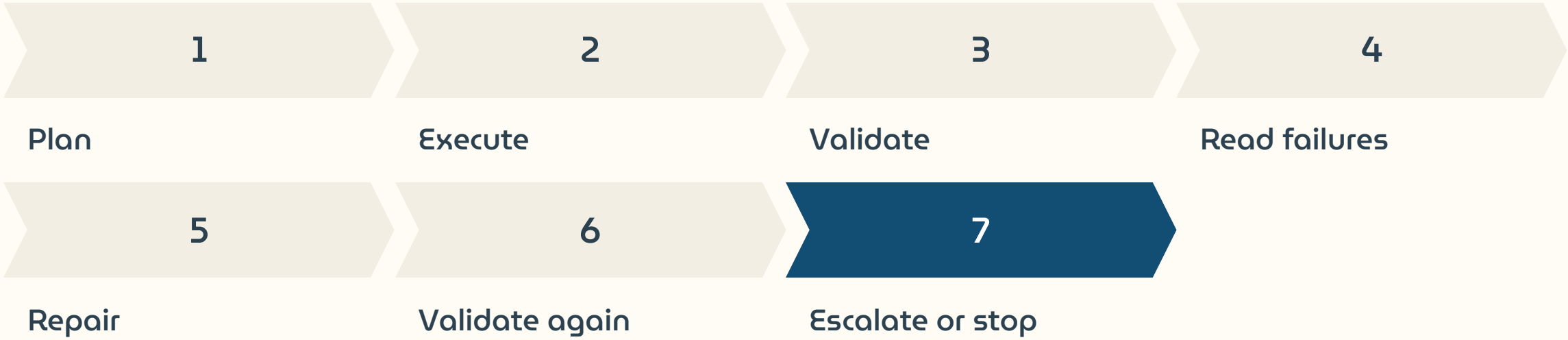
```
# Test plan: Export feature

## Scenario 3: Large account export
Steps:
1. Sign in as account with 1,200 records
2. Request CSV export, full history
3. Expect "processing" state, not a download
4. Expect email with download link

Expected: file contains 1,200 rows, oldest first
```

From intent to test plan to spec.

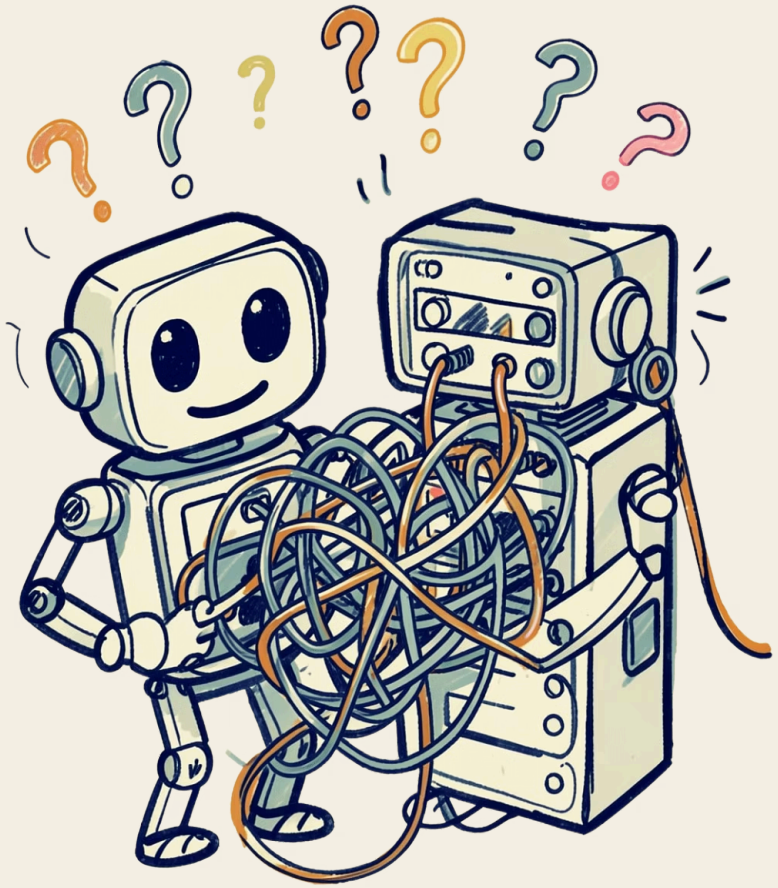
Prompt engineering optimizes a message. Loop engineering optimizes the pipeline.



stop conditions · retry caps · required test commands · fresh-context validation · human escalation · deterministic hooks

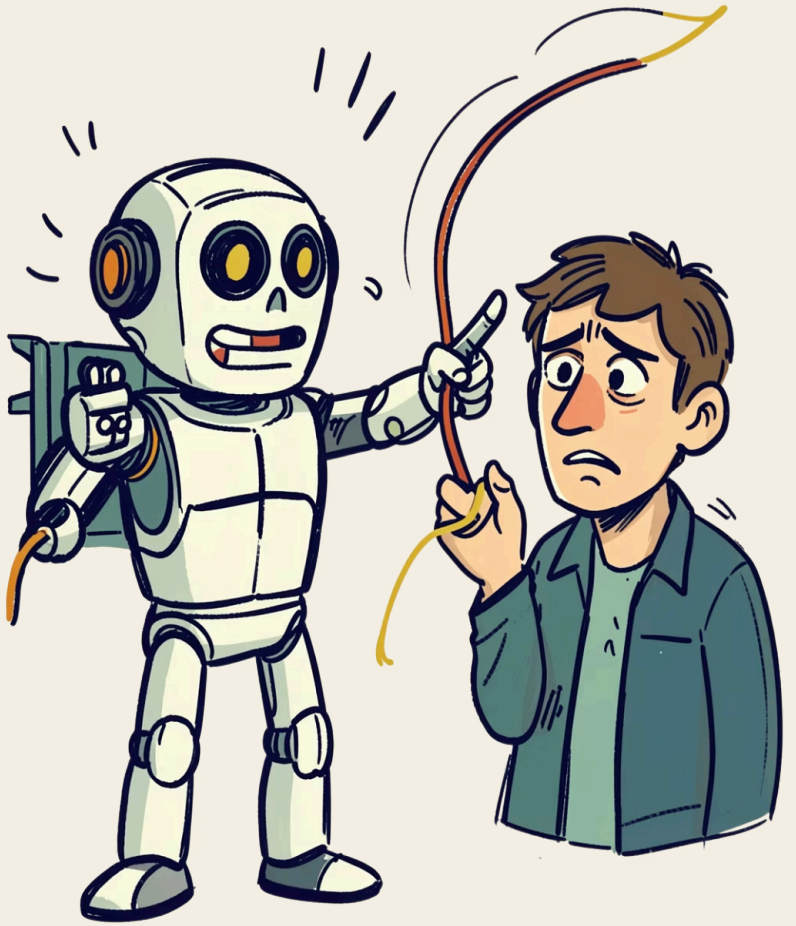
"It didn't all work."

War story 1: The undocumented deployment.

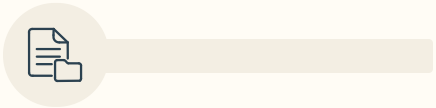


War story 2: The queue that woke up.

War story 3: When the reviewer was wrong.

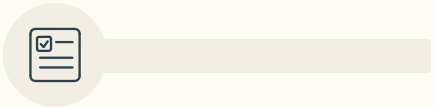


One file per week.



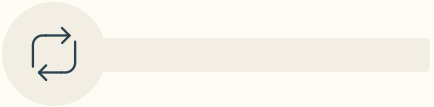
Week 1

repository instructions, an AGENTS.md



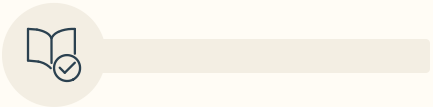
Week 2

a reviewed plan before one feature



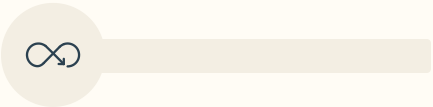
Week 3

one repeated workflow becomes a skill



Week 4

one natural-language test plan, generated into a real test



Week 5 (optional)

one loop with stop and escalation rules

"The hottest new programming language is English."

Andrej Karpathy, 2023

"We spent twenty years learning to write code."

"The next ten are about learning to direct it."

@avindrafernando

Taprobane Consulting · taprobaneconsulting.tech

Taprobane Consulting

Taprobane Consulting

Creativity and Excellence Together

@avindrafernando