Corentin Dallenogare

Agentic development harness: how it works

Full-stack and blockchain engineer · Belgium, remote
corentda@gmail.com · github.com/falcoda · linkedin.com/in/corentin-dallenogare · coval-tech.com

I build software with AI coding agents every day, mostly Claude Code. Most of that work is not prompting. It is making a codebase legible to the agent, and making its mistakes fail loudly instead of quietly. These notes describe the harness I built and maintain for that, on my agency's stack (TypeScript, React, Node.js, PostgreSQL). The repository is private because it also holds client work, so I would rather walk you through it live.

1 · CONTEXT THE AGENT READS

One source of truth, scoped by folder

A root instruction file holds the repository-wide rules. Each area has its own file, and a deeper file overrides the root for its subtree. Agent-specific files only point back to the shared ones, so Claude Code and Codex read the same rules and nothing is maintained twice.

```
AGENTS.md              repository rules: no magic values, lint green, dependency age
├─ backend/AGENTS.md   API, database, auth, validation, Docker
├─ frontend/AGENTS.md  routing, state, styling, testing
└─ linkedin/AGENTS.md  writing rules for published content
CLAUDE.md              pointers back to the AGENTS.md files
```

Every non-obvious rule carries its reason

When an agent knows the failure mode behind a rule, it applies the rule to cases nobody listed. So the rules say why:

> **Rule 0 (backend).** Never create or edit the generated database type files by hand. Their internal representation relies on exact character offsets in the SQL source, so a hand-edited file can fail silently at runtime. The agent runs the generator itself: the backend does not compile until it has.

2 · TURNING RECURRING MISTAKES INTO CHECKS

A rule an agent can ignore is a suggestion. A recurring mistake becomes a check that fails.

Recurring mistake	Written rule	Check that enforces it
Hand-written database types	Types come only from the SQL, through the generator	The build fails until the generated files exist
API shapes drifting between backend and frontend	One schema per contract. The OpenAPI document and the frontend types are derived from it, never written by hand	A CI job regenerates the contract and fails if the committed one is out of date
Transactions opened by hand in business code	Multi-statement transactions go through one helper	An ESLint `no-restricted-syntax` rule rejects them outside the database layer
Hardcoded colours in components	Colours are design tokens	Stylelint rejects hex, named and literal functional colours
Installing a package published yesterday	Seven-day minimum age for any dependency, manual installs included	Dependabot cooldown for automated updates, the written rule for the manual path

In each case the rule came first, and it was promoted to an automated check once it proved not to be enough.

3 · GATES ON EVERY CHANGE

- **On every pull request:** backend lint, tests and build, frontend lint, style lint, test typecheck, tests with coverage and build, plus the contract-drift job. The production deploy runs the test suite again before building.
- **Least privilege in CI:** the workflow token is read-only by default, so code from a pull request never inherits write access.

Gates on generated artifacts, not only on code

Agents generate documents as well as code, and a document can be wrong in ways no unit test sees. My CVs are generated from typed configuration and rendered to PDF. Once, a one-pixel overflow past the A4 height produced a blank second page that nothing caught.

The immediate fix was a print-margin tweak. The lasting one was a **delivery gate** that every generated CV must now pass before it is sent: exactly one page, at least four clickable links, a word budget, required skills present, forbidden words absent. It reads the text with `pdftotext`, which is the view an applicant-tracking parser gets, and falls back to the rendered page with an explicit warning when the tool is missing.

4 · REUSABLE SKILLS

Eleven skills, shared by Claude Code and Codex. Each one is a procedure with its guardrails, including the failures that shaped it. Among them:

- Component and store scaffolding that follows the project's conventions
- User stories written in the project's exact format, with backend and frontend notes
- A UI and UX audit that drives a real browser across three screen widths, then applies only the fixes that pass lint and tests
- A security audit

5 · PROTOCOL WORK, JUNEO (2022 TO 2026)

Four years on the Juneo ecosystem, an Avalanche-derived Layer 1:

- **Supernet Bridge:** cross-chain transfers live on five EVM chains (JUNE, BSC, Base, Avalanche, Ethereum), with lock and mint handlers and relayer voting.
- **Wallet suite on three surfaces:** web in React, mobile in React Native with biometrics, and a Chrome extension, plus a MetaMask Snap.
- **JuneoJS:** merged contributions to the TypeScript SDK the ecosystem builds on.
- **Solidity in production:** an upgradeable stablecoin-type ERC20 with pause, blacklist and role-based minting, more than a hundred wrapped JRC20 tokens, a multisig and on-chain auctions.

That work is EVM rather than Cardano or zero-knowledge systems. What carries over is SDK and developer-experience work on a live chain, where the documentation and the reference code decide what builders, and now agents, can actually do.