

Ambiguity Used to Be a One-Time Cost

Tribal Knowledge

Vague code survives because human engineers remember the context (like why the auth module split, or avoiding Friday deploys).

NO
FRIDAY
DEPLOYS

The Context Engine

Amortized Confusion

When humans are confused, they ask a teammate. They learn it once, store it in their head, and the cost drops to zero for year two.

The codebase was never just documentation. It could afford to be vague because a warm body held the missing half.

The Agent's Amnesia Tax

Your repository's maintainer just changed from a human who remembers to an agent that re-derives everything from disk.

Human Maintainer

- Memory: Stateful (accumulates context).
- Ambiguity Cost: Paid once, stored in the brain.
- Tribal Knowledge: A minor inefficiency.



AI Agent

- Memory: Stateless (starts every session cold).
- Ambiguity Cost: Paid every single session, forever.
- Tribal Knowledge: The single most expensive bug in your repo.

A messy repo a senior human navigates by memory is a repo an agent will get lost in, repeatedly, at scale.

API 1: Navigation (Directory Layout)

features/billing/engine/

Tells a true story. Billing owns this. It's domain logic. No UI imports allowed. The agent knows this before opening the file.

Vague folders are navigation endpoints that return garbage.

misc/ or utils2/ or
src/components/Dashboard.tsx

Actively lies. A dumping ground that requires the agent to read 2,000 lines just to discover if the pricing rule lives there.

わア

utils2?

API 2: Constraint (Contracts & Boundaries)

1

Implicit Contracts. Event names typed as bare strings ('user.updated'), undocumented CSV column orders, or shared storage keys.

2

Invisible Boundaries. The agent has no way to know it just broke a producer in another service because nothing in the repo explicitly stated the boundary existed.

3

Cascading Failures. An agent that edits blindly breaks things it cannot see.



Boundaries must be machine-checkable. Replace implicit tribal knowledge with explicit schemas, typed contracts, and imported constants.

API 3: Validation (Verification)

An agent alone will overfit and declare victory. It needs **one deterministic command (make verify)** to run **run** types, lint, tests, and rules.



Navigate.
(Directory Layout
tells the story).

Constrain.
(Explicit Contracts
define the rules).

Validate.
(Verification
Commands
prove success).

**Done stops being a feeling
and becomes an exit code.**

The architecture that is perfect for agents is the exact same architecture we always knew was right for humans. The agent just removed your