"AI dev dies at scale" — answered

How design crushes the failure mode.

ai-desk methodology — Hiroyuki OKINOI · 2026-05

1 The standard critique

> **Letting AI loose on a codebase is fast at first.**
> **But once it grows, AI loses context and starts breaking things.**
> **Past 50,000 lines, you're stuck.**

This critique **is correct under conventional design**.
What happens when conventional design is rejected?

2 How conventional code dies at scale

AS SIZE GROWS

- ✕ Functions call other functions → context chains
- ✕ Files depend on files → multi-file reads to edit safely
- ✕ Shared helpers (DRY) → one change ripples everywhere
- ✕ Abstraction / encapsulation → hidden info AI can't see
- ✕ Implicit assumptions → break silently

RESULT

- ✕ Exceeds AI's spotlight (~±300 lines)
- ✕ Eats the context window
- ✕ Blast radius unreadable, AI breaks things
- ✕ Tests fail, fixes cascade, snowball stalls
- ✕ Conclusion drawn: "AI has limits"

3 Cause behind the cause

AI breaks because **the design wasn't built for AI**, not because of inherent AI limits.

Conventional design principles (DRY / abstraction / encapsulation) optimise for **humans reading code**. AI needs the opposite optimisation:

- **Maximise locality** — read one place, see everything
- **Forbid sharing** — bound the blast radius of any edit
- **Be explicit** — expand inline rather than hide
- **One-file completeness** — avoid file-jump cost

This is Bible §0.0 "Cognitive Asymmetry". The same code is hard for AI and easy for humans, vice versa. **Conventional best practice is what kills AI.**

4 Strategy 1 — Heavy Function

Complete inside one function. AI reads that one function and knows everything.

Don't chop functions

Build large feature-scoped functions. Inline data transforms, branches, and side-effect triggers into one scope.

Don't fear length

200-500 line functions are fine. They just need to fit AI's spotlight. Splitting smaller increases inter-function dependency jumps.

Counter-intuitive. But **1 function = edit blast radius bounded inside it**. Other functions stay safe at scale.

5 Strategy 2 — Forbid shared helpers

No "common helpers" called from multiple functions. Don't fear duplication.

DRY (CONVENTIONAL)

- ✗ Extract identical code, reuse
- ✗ Praised as "good design"
- ✗ But: 1 change ripples to all callers
- ✗ Blast radius = O(callers)
- ✗ Larger scale, worse damage

AI-DESK: NO SHARING

- ✓ Similar code is **duplicated into each function**
- ✓ "Duplication isn't evil — hidden dependency is"
- ✓ 1 change leaves others untouched
- ✓ Blast radius = O(1)
- ✓ Damage doesn't grow with size

6 Strategy 3 — Emblem boundaries + local read

Don't physically split a giant file. Virtually partition it and let AI read locally.

- Declare sections with `// [ai_s_emblem:#layer Name]`
- `node ai-desk.js file.js skeleton` → list all emblems (dozens of lines)
- `node ai-desk.js file.js focus EmblemName` → extract just that emblem
- `node ai-desk.js file.js apply patch.js` → atomic apply (pre-flight verify + tag immutability check)

> **Even in a 100,000-line file, AI reads only the few hundred lines of the relevant emblem.**
> **Context consumption doesn't depend on file size.**

7 Strategy 4 — 4-Layer + REAL/SHADOW

Structurally separate data flow and state ownership.

4-layer one-way flow

L1 Physical → L2 Intent → L3 Logic → L4 Draw. Direction is fixed → editing X structurally determines what changes.

REAL — only one mutator

Only L3 may update `REAL_state`. Other layers are read-only. State-leak bugs become **structurally impossible**.

SHADOW is disposable

Derived values aren't stored. Compute, use, throw away. "Sync miss" bugs vanish.

Bridges expose layer crossings

Cross-layer calls tagged `// [ai_s_bridge:L3toL4]`. AI sees "this is a cognitive jump point".

8 Strategy 5 — Constraint Folding

Replace nested if/else with "enumerate all worlds → filter".

- **Conventional:** branch trees → combinatorial explosion at scale, missed test cases, AI lost in branches
- **ai-desk:** enumerate all possible worlds as a **data structure** → filter by constraints → surviving worlds = answer
- No branch tree = no AI cognitive split
- Reverse lookup possible (output → input), exhaustive testing trivial

Minimal proof: `constraint-janken.js` — 3-player rock-paper-scissors, 27 worlds, zero if statements.
Real example: `fighter-cancel.test.js` — 1920 worlds exhaustive, single file.

9 Strategy 6 — Twin + Event Sourcing

Guarantee maintainability and verifiability structurally.

Twin (verification twin)

A pure CPU function runs alongside the GPU implementation, recomputing the same input. When a bug appears, you can definitively say "this is a logic bug, not a rendering bug".

Event Sourcing

Don't overwrite state. Append the Command history as a JSON array. Each event includes the previous hash → tampering is mathematically detectable.

→ Reproduce any past state, completely audit "why did the state become this".

Even at scale, **you can trace back to the cause**.

10 Numbers the design guarantees

±300 lines
AI's read window
(per emblem, file-size independent)

O(1)
Edit blast radius
(no shared helpers)

O(change)
Verification scope
(Twin / per-goal exhaustive)

All of these are **independent of total size (file count / line count)**.

At 100,000 lines, the numbers don't change.

In conventional codebases, all of these grow with size. **ai-desk severs that relationship by design.**

Crushed by design

"AI dev dies at scale" applies

only when you keep writing the conventional way.

Redesigned for AI, there is no remaining cause to die.

Heavy Function · No shared helpers · Emblem local read · 4-Layer + REAL/SHADOW · Constraint Folding · Twin + Event Sourcing

—— once these line up, **scale is no longer a failure mode**.

github.com/AoyamaRito/ai-desk

Hiroyuki OKINOI · Aoyama Rito