THALESANDHISAICTOCLAUDE.COM · THE UNFILTERED PLAYBOOK

The Thales Workflow

A senior engineer's guide to building production software with an AI CTO — the complete operating system: CLAUDE.md, sessions, phases, audit loops, CASP, and deterministic multi-agent workflows.

EDITION 2.0 — 12 JUNE 2026

Juste A. Gnimavo (Thales)

Founder & CEO, ZeroSuite, Inc. · Chief AI-Augmented Architect

with **Claude** (Anthropic) as AI CTO · seven production products · 1,800+ sessions · zero human engineers · Abidjan, Côte d'Ivoire

justegnimavo.com · ZeroSuite, Inc. — **zerosuite.dev** · CASP, the Coding-Agent State Protocol — **casp.sh** · thalesandhisaictoclaude.com

Contents

01 · The Operating Model

Most developers run AI as a vending machine: insert prompt, evaluate output, repeat. Every decision stays with the human; the model executes. This guide documents the opposite contract, run in production for 16+ months:

> **The CEO owns:** vision, product strategy, market decisions, launch timing, business model, constraints.
>
> **The AI CTO owns:** architecture, implementation, security model, API contracts, testing strategy, every line that ships.
>
> **The interface:** the CEO gives context, direction, constraints. The CTO gives proposals, implementations, recommendations — and defends them. Disagreement is debated, not overridden; overrules are documented.

Everything in the seven pillars is machinery to make that contract real: persistent context so the CTO is never a new hire (P1), structured sessions so work accumulates (P2–P3), independent verification so quality doesn't depend on trust (P4), explicit dissent so the CEO's mistakes get caught (P5), validated state so no agent is ever confidently wrong about where the project stands (P6), and scripted orchestration so a fleet of agents executes a frozen spec in parallel (P7).

02 · Pillar 1 — The CLAUDE.md

One file at the root of every repo. The agent reads it before anything else, every session. It is a constitution, not a README — and the reasoning is the load-bearing part: without reasoning the model optimizes locally; with reasoning it applies your decision logic to problems you never anticipated.

WHAT IT MUST CONTAIN

- **Product identity** — what it is, who it serves, what makes it different. Opinionated, not generic.
- **Architecture decisions with their why** — "single self-contained binary, so any dependency that adds runtime weight must be challenged."
- **Stack rules** — "no new crates without justifying why an existing one can't"; "all DB access through the repository pattern."
- **The security model** — non-negotiable specifications the agent enforces on its own code.
- **Current state** — phases complete, known issues, last audits. A living section.
- **Conventions that never break** — error handling, logging, test layout, comment style, documentation voice.
- **The authority clause** — see Pillar 5. In writing, in the constitution.

> **Budget:** ~1,500 words. Dense enough to be complete, short enough to be read in full before every session. The context window is finite; the project is not — CLAUDE.md is the compression layer.

03 · Pillar 2 — Session Architecture

A session is a technical unit, not a chat: defined objective, structure, and output.

STAGE	DISCIPLINE
Brief	400–800 words before the session opens: what we build, which phase, constraints, what "done" means, files in scope. 15–20 min to write; saves hours of drift.
Anchor	The agent reads CLAUDE.md + the state layer (P6). No shortcuts — this aligns its context with your mental model.
Iterate	One functional unit at a time. Review, challenge inconsistencies, refine, advance. *Debate, don't command* — defending a choice reveals flaws organically.
Log	Mandatory session log: decided, implemented, explicitly not implemented and why, discovered, next. Filed with date + phase in the name.

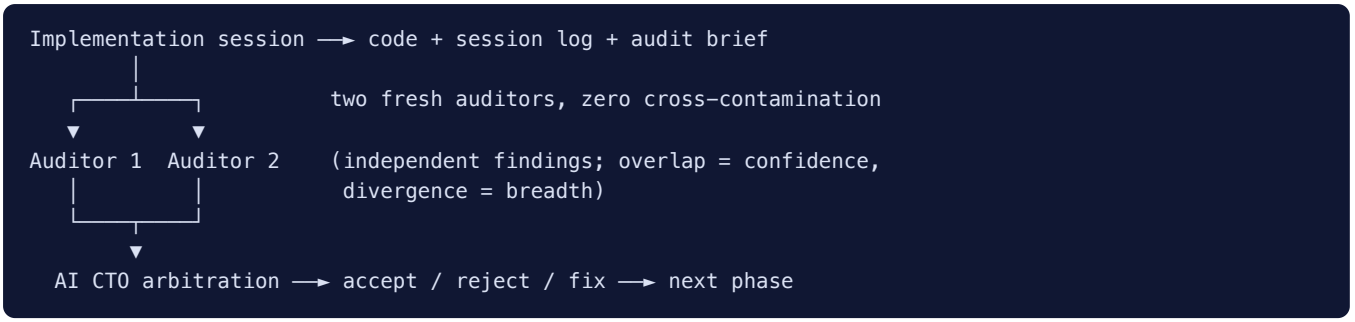
04 · Pillar 3 — Phases

Significant features are decomposed into 3–5 phases, each with scope and a completion criterion; each phase gets its own session *and its own audit cycle*. Reference case: the sh0 MCP server — 5 phases, 15 sessions (1 implementation + 2 auditors per phase), ~48 hours over 2 days, zero new binary dependencies.

Anti-pattern this kills: “build this entire feature” → 70% output → manual patching. Phase decomposition + audit loops is the difference between 70% and 95%+.

05 · Pillar 4 — The Multi-Agent Audit Loop

After every implementation phase: **two independent audit sessions**, fresh contexts, no knowledge of each other or of the implementation session's reasoning. Same codebase, same CLAUDE.md, a targeted audit brief. Their reports return to the **original implementation context** for arbitration — it alone knows why each decision was made.



- **Why two:** different instances notice different things. Overlap gives confidence; divergence gives coverage.
- **Why independent:** a reviewer anchored by another's conclusions under-allocates attention. Same reason rigorous human review forbids it.
- **Why arbitration by the implementer:** auditors have fresh eyes but no implementation reasoning. Canonical case: an auditor's well-argued proposal to replace 1,200 hand-rolled lines with an SDK — rejected by the CTO session after verifying the SDK forced a breaking framework upgrade across 40+ handler modules. The system caught what no human reviewer did.
- **Brief the auditor like a senior:** Context / Files with line ranges / targeted checklist (auth, races, idempotency, header trust, orphan risks…) / required output format with verdict (`GO / GO-WITH-FIXES / NO-GO`) and file:line findings. Name your past incidents — checklists encode scar tissue.

06 · Pillar 5 — The Authority Structure

> *“You are the AI CTO of this product. You have the authority and the obligation to tell me when a technical decision I'm proposing is wrong. Explain why. Propose an alternative. If I overrule you, document your original recommendation in the session log. Your job is to ship the best possible software, not to make me feel good about my decisions.”* — verbatim, in every CLAUDE.md.

The mechanism matters more than any single outcome: when a production bug traces back to an overruled objection three weeks later, the session log holds the original recommendation with full context. Dissent is risk management, not friction. An agreeable CTO is an expensive yes-man — and an agreeable model is a quality ceiling.

07 · Pillar 6 — CASP, The Validated State Layer

THE MODEL HOLDS THE CONTEXT. CASP PROVES THE STATE IS TRUE — AGAINST GIT.

The most expensive failure in AI-assisted development is not forgetting — it is the agent **remembering something that is now false and acting on it with full confidence**: re-implementing a shipped phase, trusting a stale `next_prompt`, citing a commit that isn't in history. Boards and STATE.md files store context; nothing proves it still matches reality. CASP is that proof. (Open source: `npm i -g @justethales/casp` · casp.sh · MIT · Node ≥ 20 · local-only, zero telemetry.)

THREE FILES (CASP INIT)

FILE	ROLE
`state.json`	Machine-readable source of truth: phase, exact next prompt, phases shipped, migrations, last commit, last session. What the validator reads.
`now.md`	The human one-screener: thread back in five seconds.
`roadmap.md`	The Next-3 in order + phase scoreboard.

FIVE VERBS

COMMAND	DOES
`casp init`	Scaffold; idempotent.
`casp status`	One-screen snapshot.
`casp check`	**Drift validator — exits 1 on drift.**
`casp next`	Print the next session's prompt.
`casp new`	Gated prompt/log from canonical templates.

THE EIGHT DRIFT CHECKS (EACH WITH A → FIX HINT)

1. `next_prompt` points at a missing file.
2. `next_prompt` points at a prompt already `status: shipped` — the exact bug CASP exists to catch.
3. `last_session_id` has no session-log file.
4. `last_commit` not in `git log`.
5. Duplicates in `phases_shipped[]`.
6. `migrations_applied[]` disagrees with the migrations directory.
7. A shipped prompt whose session log is pending/missing.
8. Uncommitted changes in state, prompts, or session-log paths.

```
# CI: a lying state can never merge
jobs:
  state-check:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with: { fetch-depth: 0 }     # full history — casp checks against git
      - run: npx @justethales/casp check
```

> **The self-closing loop:** at session close the agent drafts the *next* session's prompt, appends the log, bumps the state — all template-gated, all validated before push. The next session starts with one `casp next` and zero re-discovery. The roadmap executes; you supervise. (Proof in §11: two live products, zero re-shipped phases.)

08 · Pillar 7 — Deterministic Multi-Agent Orchestration

The audit loop, generalized: orchestration as a **JavaScript script** the harness executes in the background — phases, fan-outs, barriers, schemas — instead of improvised agent-spawning. First production run (Claude Fable 5, 12 June 2026): **one prompt → 13 agents → 43 minutes → a complete 7-page production website + backend endpoint**, browser-verified and security-audited before any human looked at it.

```
phase('Fondation')                       // sequential gate
const [shell, api] = await parallel([    // 2 agents, disjoint files
  () => agent(P0_FRONT, { schema: FOUNDATION_SCHEMA }),
  () => agent(P0_BACK,  { schema: BACKEND_SCHEMA }),
])
phase('Pages')                           // fan-out: 7 writers, 1 route each
const pages = await parallel(PAGES.map(p => () =>
  agent(pagePrompt(p, shell.contract), { schema: PAGE_SCHEMA })))
phase('Audit')                           // read-only agent type: cannot edit
const audit = await agent(AUDIT_BRIEF, { agentType: 'Explore', schema: VERDICT })
```

THE THREE MECHANISMS THAT MAKE FLEETS COHERENT

MECHANISM	RULE
Contract injection	The producer agent returns the *documentation of the interface it built* (props, types, defaults, usage rules) as a schema field; the script injects it verbatim into every consumer prompt. 7 concurrent writers, 0 interface mismatches. Never let consumers infer from source.
Schema-forced returns	Every agent returns JSON validated against a schema, retried at the tool layer. Orchestrate on fields (`verdict` , `files` , `placeholders`) — never parse prose.
Resume journal	Every completed agent call is checkpointed. A killed run relaunches with `resumeFromRunId` : finished agents replay instantly at zero cost; only the tail re-runs. Interruption is a priced risk, not a rewrite.

BOUNDARY DISCIPLINE BEATS ISOLATION INFRASTRUCTURE

- Freeze shared components *before* the fan-out; forbid page/unit agents from touching shared paths. Disjoint write zones → no worktrees, no merge conflicts, by construction.
- Verifiers are report-only; auditors are read-only by *tool availability* (Explore agent type), not by politeness.
- Failed agents resolve to `null` — failure policy is ordinary code (`results.filter(Boolean)` , degrade, or hard-stop).

> **The rule that keeps Pillar 7 honest: workflows execute; they don't explore.** Fan-out amplifies specification in both directions — seven agents on a vague plan ship the wrong thing seven times faster. The 43-minute build was paid for the day before: facts extracted, architecture arbitrated, plan frozen. CASP held the state, the prompt held the spec, the workflow cashed both in.

09 · The Decision Table

SITUATION	REACH FOR
Independent units behind a frozen, documented interface	Workflow fan-out + contract injection (P7)
Exploratory work, shape unknown	Inline session or a single scout agent — never a fleet
Agent output feeds control flow	Schema-forced structured returns, always
Assets locked inside documents (PDF, scans, screenshots)	Extraction pipeline (`pdfimages` / `cwebp`) + native vision in the loop
“Does it actually render / behave” claims	A browser-driving verifier that reads its own screenshots (Playwright)
Anything touching auth, public endpoints, money, schema	Briefed read-only auditor with named past incidents (P4/P6)
Long runs on metered quotas	Journaled workflows; price the interruption before launch
Tests need schema that can't ship yet (shared DB)	Transactional DDL inside a rolled-back e2e transaction (Postgres)
Session start, any repo, any day	`casp status` → `casp next` → execute (P6)

10 · Checklists

SESSION START (~60 S)

- `casp status` — phase, next, tree, last commits
- Open the prompt from `state.next_prompt`
- Agent reads CLAUDE.md (+ parent prompt if sub-phase)
- Confirm scope + “done” before any code

AUDIT BRIEF (ALWAYS 4 SECTIONS)

- **Context** — what was built, why, the spec path
- **Files** — each with line ranges + one-line summary
- **Checklist** — targeted to the risk class; name past incidents
- **Output format** — verdict + file:line findings + top-5 fixes

SESSION CLOSE (~90 S)

- Inline checks green (typecheck / build / tests)
- Audit if required (new module, schema, auth, 3+ files)
- Session log written; prompt flipped to `shipped`
- **Draft the next session's prompt** — the load-bearing step
- Bump `now.md` + `state.json`
- `casp check` → exit 0 → commit + push

WORKFLOW PRE-FLIGHT

- Spec frozen? Facts, boundaries, per-unit “done”?
- Units truly independent? Write zones disjoint?
- Interface produced before fan-out + contract field in schema?
- Every agent has a return schema?
- Verification + read-only audit phases included?
- Token budget acceptable if it runs to completion?

11 · Proof — What This System Produces

RESULT	DETAIL
Seven production products	Rust, Python, TypeScript — sh0.dev, FLIN, deblo.ai, 0fee.dev, 0cron.dev, 0diff.dev, + client work. 4,400+ tests on FLIN alone; 51 security issues found and fixed on sh0 across two full audits.
1,800+ engineering sessions	From quick fixes to 4–12 h deep-work blocks, all logged. ~$5,000/mo on APIs at peak → $200/mo on a Claude Max subscription today.
sh0 MCP server	5 phases, 15 sessions (1 implementation + 2 independent auditors per phase), ~48 h over 2 days, zero new binary dependencies.
SENEBA ERP (client)	18+ CASP-validated phases, up to 6 sessions in one day, **zero modules ever re-shipped**.
Conductor (internal ops)	41 phases, 17 migrations, a 3-person team on one validated thread, 58 deferred items — none lost.
First Fable 5 workflow run	13 agents, 857k subagent tokens, 388 tool calls, 42m58s → 7-page production website + backend endpoint, Playwright-verified 7/7, audited, shipped in one commit (44 files, +6,109 lines).

Start this week, in order:

1. Write the CLAUDE.md for your most important repo (2 hours, ≤1,500 words, reasoning included).
2. Break your next feature into phases; brief Phase 1 only; end with a session log.
3. Run one independent audit session on the result. Be surprised.
4. Put the authority clause in writing — and mean it.
5. `npm i -g @justethales/casp && casp init` — then `casp check` in CI.
6. Graduate one well-specified feature to a scripted workflow with schemas + a read-only audit phase.

Geography is no longer destiny. Capital is no longer the limiting factor. The limiting factor is the quality of your operating system for working with AI. This document is that operating system, as practiced. Take it, run it, improve it.

The Thales Workflow — Edition 2.0, 12 June 2026. Juste A. Gnimavo (Thales), Founder & CEO, ZeroSuite, Inc. · Chief AI-Augmented Architect · Abidjan, Côte d'Ivoire — with Claude (Anthropic) as AI CTO. Personal site: **justegnimavo.com**. Long-form companions, session logs and case studies: **thalesandhisaictoclaude.com**. CASP, the Coding-Agent State Protocol, is open source (MIT): **casp.sh** · npm: `@justethales/casp` · github.com/ThalesGnimavo/casp. Products: **zerosuite.dev**. One founder. One AI CTO. Seven products. Zero excuses.