

The Thales Workflow

A senior engineer's guide to building production software with an AI CTO — now opening with the step-by-step playbook from empty folder to launched product, backed by the complete operating system: CLAUDE.md, sessions, phases, audit loops, CASP, deterministic multi-agent workflows, the self-improving build loop, and the fleet with its arbitration guard.

EDITION 5.0 — 21 AUGUST 2026

- P1 The CLAUDE.md — the CTO's constitution
- P2 Session architecture — brief, anchor, iterate, log
- P3 Phase-based feature development
- P4 The multi-agent audit loop
- P5 The authority structure — Claude can say no
- P6 CASP — the validated state layer
- P7 Deterministic multi-agent orchestration
- P8 The self-improving build loop
- P9 The fleet — named sessions, one writer, arbitration first



Juste Thales Gnimavo

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

© 2026 ZeroSuite, Inc. · Licensed CC BY-NC-ND 4.0 · Full notice on the last page

Contents

01	The operating model — CEO owns intent, AI CTO owns implementation
02	The Playbook I — stages 0–4: equip, decide solo, first ten minutes, design, first brief
03	The Playbook II — stages 5–9: rhythm, the day, verification, fleet, launch
04	Pillar 1 — CLAUDE.md, the constitution every session reads first
05	Pillar 2 — Session architecture, the unit of work
06	Pillar 3 — Phases, decomposition with completion criteria
07	Pillar 4 — The audit loop, independent reviewers + informed arbiter
08	Pillar 5 — Authority, the agent's documented right to refuse
09	Pillar 6 — CASP, three files, five verbs, one git-grounded validator
10	Pillar 7 — Workflows, scripted fleets of agents (anatomy + patterns)
11	Pillar 8 — The build loop: beats, verifier, compounding state, routines
12	Pillar 9 — The fleet: the launcher, the arbitration, one writer
13	The decision table — which tool for which job
14	Checklists — session, audit, workflow, loop & fleet pre-flight
15	Proof — the numbers this system produces
16	The author — the record behind the method, and how to work with him

New in Edition 5.0 (21 August 2026) — The Playbook: ten stages from an empty folder to a launched product, with the exact commands and the exact first messages — the answer to the question readers kept asking: *"I believe the system. What do I type, in what order?"* Two new pages (§02–§03), written for the developer on day one and the veteran on year twenty alike. The pillars are unchanged below — they remain the **why** behind every step; the daily slash-command set is documented in the companion article on thalesandhisaictoclaude.com. The guide's source stays version-controlled: every edition is an edit

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 17+ 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 nine 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), scripted orchestration so a fleet of agents executes a frozen spec in parallel (P7), a self-improving loop so the roadmap keeps executing, with guardrails, when nobody is at the keyboard (P8) — and, when several full sessions run at once, an **arbitration that decides the shape of the work before any of them writes** (P9).

02 · The Playbook I – From Empty Folder (Stages 0–4)

NEW IN 5.0 – WHAT TO TYPE, IN ORDER, FROM A FOLDER THAT DOES NOT EXIST YET TO A FIRST SLICE THAT RUNS

The pillars are the why. These two pages are the what: ten stages from zero to launch, every one measured on real work.

Prerequisites: git basics, a terminal, a Claude Code subscription. That is the entire list.

STAGE 0 – EQUIP YOURSELF (ONCE PER MACHINE, ABOUT AN HOUR)

Install Claude Code, log in, set the model explicitly (`/model`) — never inherited by accident. Then the state layer:

```
npm i -g @justethales/casp
SKILLS="$(npm root -g)/@justethales/casp/skills"
mkdir -p ~/.claude/skills && cp -r "$SKILLS/casp" "$SKILLS/next" ~/.claude/skills/
```

CASP is the one check in the whole loop that is **not** the model checking itself: deterministic, local-only, zero telemetry. The daily slash-command set — context, naming, permissions, loops, security review — is documented in the companion article on thalesandhisaictoclaude.com.

STAGE 1 – THE DECISION BEFORE THE FIRST LINE OF CODE (FIVE MINUTES)

Start solo. Always. A skeleton has no independent surfaces — every parallel session lands in the same three files, at N times the budget. Parallelism is a per-work-item decision taken once the code has shape (Stage 8), never a default, never on day one. The first solo session is what *creates* the lanes.

STAGE 2 – THE FIRST TEN MINUTES OF THE REPOSITORY

```
mkdir my-project && cd my-project && git init
casp init          # state.json · now.md · roadmap.md + first session prompt
casp install-hook  # casp check runs on every git push, from now on
casp check         # green out of the box
git add -A && git commit -m "chore: scaffold the state layer"
```

`casp install-hook` is the step people skip and regret: with it, a state file that lies can never reach your remote. Commit before the agent starts — the validator gets a history to compare against, you get a reset point.

STAGE 3 – DESIGN BEFORE CODE (GREENFIELD PRODUCTS)

Claude Design goes first: a token system, typed component contracts, clickable UI kits per surface. The design system is a contract exactly like an API — hand it to the engineering agent and the build session becomes a faithful port. For a CLI, a library, an API-only product: skip without guilt.

STAGE 4 – THE FIRST MESSAGE TO THE AGENT

```
New project. The state layer is scaffolded (casp/) and the pre-push gate is
installed – read casp/README.md before touching anything.
What we are building: <2–3 sentences: what it does, for whom, and the one
constraint that rules out the obvious approach.>
This session, solo: pick the stack and defend the choice, lay out the
structure, fill casp/state.json + casp/roadmap.md with a real first slice.
Write the code for that slice only. Do not parallelize anything yet.
```

Three deliberate moves: **name the constraint** (or the agent picks the most conventional stack), **scope to one slice** (or you get a plausible skeleton nobody can verify), and **close the parallelism question** before the agent proposes five terminals. This session also writes the CLAUDE.md — the constitution of Pillar 1, authority clause included.

03 · The Playbook II – Rhythm, Day, Verification, Launch (Stages 5–9)

STAGE 5 – THE RHYTHM OF EVERY SESSION AFTER THAT

BEAT	COMMAND	DISCIPLINE
Start	<code>/next · casp next</code>	Prints the next session's prompt from the recorded plan. Refuses to start on a drifted state — a bad session never begins.
Work	—	The agent implements the slice, and nothing else.
Close	<code>casp ship · casp close</code>	Phase shipped, log wired, state bumped to HEAD. Neither touches git; you own the commit.
Gate	<code>casp check</code>	Exit 0 clean, exit 1 drift — automatic on push once the hook is installed.

Two habits worth more than they look: write the **next** session's prompt before closing the current one, and let the state bump be its own commit — one commit is what changed, the next is what it means.

STAGE 6 – RUNNING YOUR DAY

One session or several? The honest default is one. While the agent works, **do nothing** — the only messages worth sending mid-flight are decisions, unblocks and corrections. Read every closing report as a claim, not a result: a green result with no pasted output means nobody looked; a screenshot proves something rendered, never *what*; counts drift. The one-line question that pays every day: "*which of these did you verify, and which did you infer?*" Close the day with pathspec commits (`git commit <paths> — never -a`), one consolidated log with raw output in it, and a state pointer a fresh session can start alone tomorrow.

STAGE 7 – VERIFICATION, LAYERED

- **The deterministic floor** — typecheck, build, tests, `casp check`. Exit codes, not opinions.
- **A read-only verify agent** — runs every check in the background, writes a report, cannot edit source.
- **The audit loop (Pillar 4)** — two fresh auditors per phase; arbitration by the informed implementer.
- **A security review** — on anything touching auth, money or user data, before it ships.
- **Visual verification you run yourself** — a headless browser at 390 px, screenshots actually read.

STAGE 8 – THE FLEET, ONLY WHEN IT STOPS BEING PREMATURE

Two conditions, both required: lanes you can write down (owned, disjoint directories), and work items that are genuinely independent. **N sessions cost N quotas**, and nothing measured says a fleet is faster — what it buys is **contradiction**: someone positioned to refuse your order. Default shape: one writer plus adversarial read-only reviewers. The full machinery, the launcher and the arbitration guard: Pillar 9.

STAGE 9 – LAUNCH: A CHECKLIST, NOT A MOOD

- ▶ Deterministic gates green — with the raw output pasted, not the adjective "clean"
- ▶ One deep audit over everything since the last one: adversarial agents, the full e2e battery, the security review
- ▶ Every number a product page claims, sourced where a validator can reach it
- ▶ State closed honestly: the roadmap points at the first post-launch slice; deferrals recorded, none lost
- ▶ Deploy, then verify the deployment — the real URLs, from a machine that is not yours

The acceptance test that matters: come back after two weeks away, run `casp status`, and start working without reading the diff. If you cannot, the state layer is being filled in as a formality — and a formality passes the gate while telling you nothing.

04 · 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.

05 · 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.

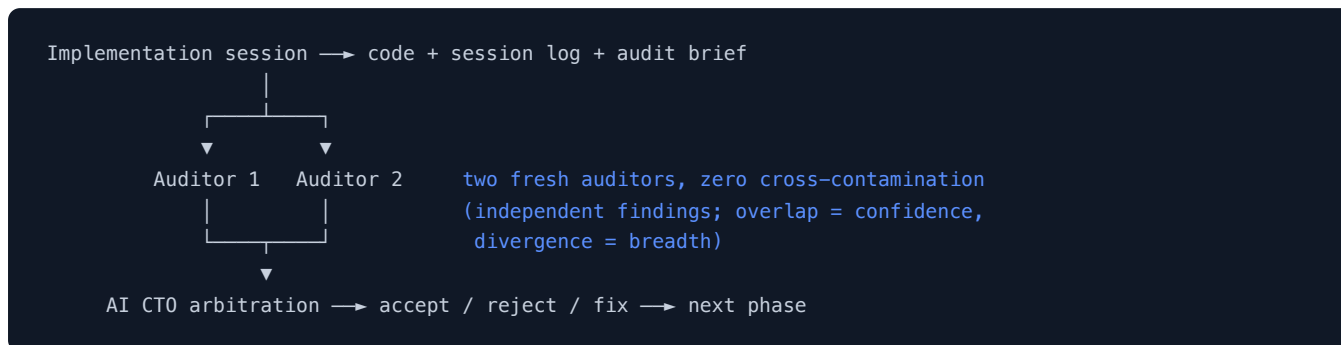
06 · Pillar 3 – Phases

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%+.

07 · 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 (G0 / G0-WITH-FIXES / N0-G0) and file:line findings. Name your past incidents — checklists encode scar tissue.

08 · 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.

09 · 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)

- `next_prompt` points at a missing file.
 - `next_prompt` points at a prompt already `status: shipped` — the exact bug CASP exists to catch.
 - `last_session_id` has no session-log file.
 - `last_commit` not in `git log`.
- Duplicates in `phases_shipped[]`.
 - `migrations_applied[]` disagrees with the migrations directory.
 - A shipped prompt whose session log is pending/missing.
 - 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 §15: two live products, zero re-shipped phases.)

10 · 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.

11 · Pillar 8 – The Self-Improving Build Loop

Pillars 4 and 7 still assume a human opens the session. Pillar 8 removes that assumption — carefully. First installed end to end on the day-zero session of senndo (8 July 2026), our newest product, before its first line of code. The loop is a contract, not a vibe:

```
TRIGGER → /loop (idle) or a scheduled cloud routine
SCOPE   → one task from progress.md, on branch claude/<task-id>
ACTION  → maker implements → verifier judges → PR if green
BUDGET  → 1 task per beat · spend cap per run · max 3 subagents/beat
STOP    → verifier PASS (→ PR) · 2 consecutive FAILs (→ ESCALATED) · budget hit
REPORT  → append progress.md + write STATE.md + push + message the founder's phone
```

THE FIVE COMPONENTS

COMPONENT	DISCIPLINE
Task queue	<code>progress.md</code> : ordered, dependency-first; every task points at acceptance criteria written as testable stop conditions an independent grader can verify. "Done" is never a feeling.
Verifier subagent	Read-only tools, its own charter: "you didn't write this code and have no stake in it passing." Runs the make gates, demands new tests per task, vision-checks UI against the design reference, runs an adversarial pass on money paths. The maker never grades itself.
Compounding state	<code>STATE.md</code> written at the end of every beat, success or failure: verified facts (provenance required), distilled rules, open failures, confirmed anti-patterns. Process rules that survive two beats get promoted into the loop contract itself.
Circuit breaker	Two consecutive FAILs: roll back the branch, write the blocker to ESCALATED, stop. No wandering to adjacent tasks. Product decisions (pricing, cash, scope) escalate by contract — the loop never guesses on business.
Graduated autonomy	First beats run read-only — summaries of what the loop would do — until task selection and verdict quality earn write access. Only then does the trigger move to a scheduled routine: beats with the laptop closed, every report pushed and messaged to a phone.

The safety argument is structural, not behavioral: every unattended path terminates in one of three mechanical outcomes — a PR a human reviews, a named blocker, or a tripped breaker. There is no fourth path where the loop quietly redefines its own mission. Under it all sits the deterministic floor: property tests on every money path and `casp check` at the push boundary — the checks with exit codes, not opinions.

12 · Pillar 9 – The Fleet, and the Arbitration That Comes First

ADDED IN 4.0 – SEVERAL FULL SESSIONS ON ONE CODEBASE, AND THE DECISION THAT PRECEDES THEM

Pillar 7 runs many *agents* inside one session. Pillar 9 runs several full *sessions* in parallel on one codebase — each a named terminal tab you can read, talk to, and stop: a controller (`cto-<project>`) that delegates and does not code, and workers launched with their brief already loaded. Everything below is what our first full fleet days (mid-August 2026) measured — including what failed.

THE OPENING MOVE – A LAUNCHER THAT REFUSES

Every session starts with one word: `cto <project>` — resolve the repository, name the session, print the model; the banner ends with the expected first gesture: `/cto` , the skill that renders the arbitration. Three refusals, each born from a measured defect:

IT REFUSES	BECAUSE
Umbrella directories	From a container folder, <code>git rev-parse</code> silently climbs to a parent repository — the session reads the parent's state believing it is its own. Refused, with the list of real repositories beneath.
Ambiguous names	Two projects match the name given. Opening the wrong one silently is the most expensive failure a launcher can produce. Refused, with the candidate list — never guessed.
Silent model inheritance	A session inheriting the machine's default model produces cost estimates about a model that is not executing. No test fails on that. The model is explicit, and displayed.

THE ARBITRATION – RECORDED, IN BOTH DIRECTIONS

Before any substantial work: **solo or fleet**, decided explicitly, recorded with a reason. In both directions — the controller may propose a fleet nobody asked for, and it may **refuse or shrink** a fleet the CEO requested when its measurements say the shape will not hold. The execution skill refuses to start when no recorded arbitration covers the phase; the escape hatch demands a written reason. The guard lives in the skill, **deliberately not in the deterministic gate**: everything the gate refuses is falsifiable against git, and "a human decided" is not. A checkbox among proofs contaminates the proofs.

What a fleet is actually for — the honest claim. Across three trials on two repositories, nothing demonstrated that a fleet is faster, and this guide will not claim it. What the trials demonstrated is that a fleet **contradicts**: workers refused the controller's premises, found the controller's own bad commit, flagged a lying tool. A solo session has nobody positioned to refuse its own order. That prices the default shape: **one writer plus adversarial read-only reviewers** — contradiction by construction, write collisions impossible. A second writer requires a written reason.

THE QUESTION THAT CLOSES AN ARBITRATION IN FIVE MINUTES

Are the project's gates **isolable per session** — or does e2e tear down a shared stack on fixed ports, against one shared test database, into one shared build directory? If not isolable, more than one writer is mechanically excluded: two sessions will not produce a visible conflict, they will produce red gates attributed to the diff. Per-project property; measure it, don't assume it.

TWO FAILURE MODES UNIQUE TO PARALLEL WORK – INVISIBLE TO CODE REVIEW

- **Stale belief.** A long session is never wrong about its own work; it is wrong about everyone else's, and the gap grows with its age. Shared state is re-verified against the remote before reasoning — never recalled.
- **Two-party false certainty.** Two sessions agree, and the agreement manufactures confidence nobody verified. Fix: on any agreement, verifying the premise is assigned **by name** to one of the two, in the message that seals it.

And the discipline that pays for it: context. On a ten-hour measured session, whole-file reads were **three quarters** of the avoidable token waste — the heavy builds cost almost nothing, because they wrote to files re-read with bounded greps. Read by range, edit surgically, bound every output — and put those rules **verbatim in every worker's brief**. Order of magnitude: a six-session fleet costs ~20× more than a full day of command outputs.

13 · 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
Pre-built plans/prototypes from other AI surfaces	Parallel read-only audit agents + a written arbitration file — before any code trusts any document (P8, senndo day zero)
Agent output feeds control flow	Schema-forced structured returns, always
Assets locked inside documents (PDF, scans, screenshots)	Extraction pipeline (<code>pdfimages</code> / <code>cwebp</code>) + 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)
Work that should proceed without you	The build loop: queue + beats + verifier + breaker, read-only first (P8)
Anything touching money	Property-tested invariants written into acceptance criteria before UI exists (P8)
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)
An empty repo you'll touch across many sessions	State protocol on day zero, while installing it is free (P6)
Session start, any repo, any day	<code>casp status</code> → <code>casp next</code> → execute (P6)
Opening any substantial piece of work	The arbitration: solo or fleet, recorded with a reason — and an execution command that refuses without it (P9)
A fleet temptation	One writer + adversarial read-only reviewers; a second writer needs a written reason (P9)
Gates on fixed ports, one shared test DB, a common build directory	Measure isolability first — if the gates are shared, the multi-writer question is already answered (P9)

14 · 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

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?

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

LOOP PRE-FLIGHT (BEFORE THE FIRST BEAT)

- ▶ Queue ordered by dependency, criteria testable per task?
- ▶ Verifier charter separate from the maker, gates named?
- ▶ Circuit breaker + budget cap set before beat one?
- ▶ Escalation channel proven live (your phone, not a terminal)?
- ▶ First beats read-only until verdict quality is judged?
- ▶ Product decisions listed as escalate-only, never guessed?

FLEET / ARBITRATION PRE-FLIGHT — ADDED IN 4.0

- ▶ Arbitration recorded for this exact phase — solo or fleet, with the reason?
- ▶ Gates isolable per session — ports, test DB, build directory?
- ▶ Lanes written out as owned, disjoint directories?
- ▶ Shared files (state, logs, lockfiles, i18n) assigned to one writer?
- ▶ Context-discipline rules cited verbatim in every worker brief?
- ▶ Commits by pathspec everywhere; shared symbols committed before their consumers?
- ▶ Model explicit and displayed for every session launched?

15 · 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.
KASSIA 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).
senndo day zero	Newest product, 8 July 2026: domain registered in the morning; by night — a validated repo, an 11-decision arbitration of three AI surfaces' pre-work, a 28-task queue, the full Pillar 8 loop wired, first <code>casp check</code> green. Zero lines of product code, on purpose.
First fleet days	Mid-August 2026, three trials on two repositories. The measured value was contradiction, not speed: workers refused the controller's premises, surfaced the controller's own bad commit, flagged a lying tool — the corrections flowed upward. One requested fleet was refused and shrunk to a single writer because the gates were not isolable. No speed claim is made: nothing measured supports one.

START THIS WEEK — THE SHORTEST ON-RAMP

- 1
- Tonight: Stage 0 + Stage 2 (§02) — an hour for the toolkit, ten minutes for your most active repository.
- 2
- Your next session: run it as Stage 5's four beats, and close it with a session log and a green `casp check`.
- 3
- Your next feature: phases, then one independent audit session on the result (Stage 7). Be surprised.
- 4
- Then let failure pull you forward: the first push the gate blocks earns the audit loop; the first time the serial queue blocks you, read Stage 8 the way it is written — as a cost, with a reason.

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 5.0, 21 August 2026. Author, corporate record, engagement terms and the full copyright and licensing notice: next page.

16 · The Author – the record behind the method

Juste Thales Gnimavo is the founder and chief executive of **ZeroSuite, Inc.**, a Delaware C-corporation he owns outright and operates from Abidjan, Côte d'Ivoire. He is the only human engineer across its product line.

He did not arrive at this method through AI. He arrived at it through operations. Two decades of infrastructure came first: **300+ servers** deployed and kept alive, **70+ applications** in production, **30+ payment gateways** integrated, thousands of sites hosted through **TPEcloud** and **Workcloud** — two companies he still leads. That is where the discipline in this guide comes from. The gates, the audit loop, the refusal to trust a green report: none of those are AI practices. They are what you learn keeping other people's systems up, at night, alone, in a market where a failed deploy is not an incident ticket but a lost client.

In January 2026 he appointed Claude Chief Technology Officer of ZeroSuite. The title describes the actual scope: every architecture decision in the seven products has been argued, and regularly lost, inside the sessions this document describes. The method was not designed and then applied — it was extracted from what survived contact with production, and published as it was learned, failures included.

THE RECORD

DIMENSION	VERIFIABLE POSITION
Corporate	ZeroSuite, Inc. — Delaware C-corporation, 100 % founder-owned, D-U-N-S 144972248. Also founder of TPEcloud (West Africa) and Workcloud (UK & West Africa). Bootstrapped: no venture capital raised to date, no outside shareholder.
Delivery	Seven production products across Rust, Python and TypeScript — sh0.dev, FLIN, deblo.ai, 0fee.dev, 0cron.dev, 0diff.dev, senndo.com — plus client systems. Zero human engineers hired.
Practice	1,800+ logged sessions. 4,400+ tests on the FLIN compiler alone. 51 security issues found and fixed on sh0 across two independent audits.
Client delivery	KASSIA ERP — 18+ phases validated against a deterministic gate, up to six sessions in a day, zero modules ever re-shipped. Conductor — 41 phases, 17 migrations, 58 deferred items, none lost.
Open source	CASP, the Coding-Agent State Protocol — MIT. casp.sh · npm @justethales/casp
In the open	The full method in English, French and Spanish, in real time — logs, arbitrations and failures alike. Nothing here is a claim without a dated log behind it.

WORKING TOGETHER

- 1 **Advisory and installation.** For a CTO, VP of Engineering or founder whose teams get demos out of AI assistance but not shippable software. Short and specific: install this operating system in your repository, adapt the gates to your stack and your CI, run the loops with your engineers until they hold without me, then leave. Weeks, not a retainer.
- 2 **Investment.** Bootstrapped, single-owner, no headcount burn by construction — the cost structure is compute and infrastructure, not salaries. Capital here buys distribution, not runway to hire. Cap table and product economics on request.
- 3 **Build.** Systems delivered under this method, gates and audit trail included — the client owns the repository, the logs and the arbitration record, not just the artefact.

hello@zerosuite.dev · +1 302 621 0207 · justegnimavo.com · zerosuite.dev · thalesandhisaictoclaude.com

Copyright, licence and authorship

© 2026 ZeroSuite, Inc. — 1130 Ogletown Rd, Ste 2, Newark, DE 19711, United States. A Delaware C-corporation. All rights reserved except as licensed below.

Juste Thales Gnimavo asserts his right to be identified as the author of this work. *The Thales Workflow, CASP — Coding-Agent State Protocol, the Nine Pillars, the four beats, the CTO controller* and ZeroSuite are marks of ZeroSuite, Inc.

LICENCE — CREATIVE COMMONS BY-NC-ND 4.0 INTERNATIONAL

You **may** copy and redistribute this document **in full**, on any medium and by any means, provided you keep it intact, credit the author, and make no commercial use of it.

You **may not** publish an adapted, rewritten, re-titled, translated, excerpted-and-restyled or AI-paraphrased version of this document, whether or not the source is named. Doing so terminates the licence automatically and leaves the act unlicensed.

creativecommons.org/licenses/by-nc-nd/4.0

THE METHOD IS YOURS; THE AUTHORSHIP IS NOT

Copyright protects this *text*, not the *practice* it describes — and the distinction is deliberate. Run this workflow. Adapt it to your stack. Teach it to your team. Build a business on it. No permission is required and none is asked for; that is the point of publishing it.

What is granted in no jurisdiction is **presenting this system, its structure or its named components as your own creation**. Under the Berne Convention and Ivorian / OAPI law, the author's right of paternity is **perpetual, inalienable and imprescriptible**: it does not lapse, it cannot be transferred, sold or waived, and it is not acquired by republishing the work under another name.

ANTERIORITY OF THE WORK

EDITION	DATE AND PUBLIC RECORD
Edition 5.0	21 August 2026 — the Playbook, ten stages from empty folder to launched product
Edition 4.0	17 August 2026 — the fleet, its launcher and its arbitration guard
Edition 3.0	8 July 2026 — CASP, deterministic workflows, the self-improving build loop
Editions 2.0 · 1.0	First publication of the nine-pillar system
Continuous record	Every claim in this guide is backed by a dated session log published in English, French and Spanish at thalesandhisaictoclaude.com , and by the version history of this document's own source.

HOW TO CITE THIS WORK

Gnimavo, Juste Thales. *The Thales Workflow*, Edition 5.0. ZeroSuite, Inc., Abidjan / Newark, DE, 21 August 2026. thalesandhisaictoclaude.com/go/workflow-guide

When you use the method rather than the document: "*Based on The Thales Workflow by Juste Thales Gnimavo — thalesandhisaictoclaude.com.*" Machine reading and AI ingestion of this document are permitted; attribution remains required of any system that reproduces it.

CONTACT

Commercial licence, training use, corporate redistribution, or permission beyond this notice:
hello@zerosuite.dev · +1 302 621 0207
ZeroSuite, Inc. legal centre: zerosuite.dev/en/legal.html · Author: justegnimavo.com