INTERNAL · ENGINEERING REVIEW

AI-Assisted Development.

The Starter Template.

What works. What doesn't. What we'd do differently.

PRESENTED BY
Engineering Team

FORMAT
Field report · ~25 min

SCOPE
Backend · Frontend · QA

01 · WHERE WE ARE

AI moved into the IDE.

It writes code. It reads the codebase. It runs commands.

Not a tool you open. A teammate you brief.

THEN

Autocomplete

Suggestions inside a line of code.

→

NOW

Agents

Whole features. End to end. In a session.

→

NEXT

Pipelines

Multi-agent workflows shipping to prod.

02 · OUR TOOL

Claude Code.

An AI engineer in the terminal.

Reads your repo. Writes code. Executes commands. Holds context across files.

READS

Your codebase

Imports, types, conventions. Patterns are inferred.

WRITES

Real code

Production-grade. Not snippets. Not autocomplete.

EXECUTES

Commands

Tests. Migrations. Git. Whatever's in your shell.

03 · WHAT'S AT STAKE

The promise is real.
The catch is structural.

60 – 65%

PROVEN OUTPUT LIFT
WHEN HANDLED RIGHT

SET UP WRONG

You burn tokens.
You ship bugs.
You re-explain the same stack every Monday.

SET UP RIGHT

Patterns persist.
Decisions survive resets.
The team levels up — not just the AI.

THIS DECK IS WHAT WE LEARNED SETTING IT UP.

ACT 01 · OUR STORY · 01 OF 04

How it started for us.

We adopted Claude Code across backend, frontend, and QA.
We moved fast. **Then we hit walls.**

MONTH 1 – 2

Adoption.

Everyone got access. Everyone got faster — at first.

MONTH 3 – 4

The walls.

Auto-accepts shipped bugs. Token bills spiked. Quality wobbled.

MONTH 5 – 6

The system.

We stopped reacting and started structuring. This deck is the result.

ACT 01 · OUR STORY · 02 OF 04

Six mistakes you don't have to make.

01
AUTO-ACCEPT

Trusting first drafts.

A Stripe integration shipped with bugs nobody caught.

02
VAGUE PROMPTS

Asking too broadly.

Stable code got rewritten when nobody asked for it.

03
DEAD CONTEXT

Carrying old chats.

Long threads cost more and produce worse output.

04
RESUME TRAP

Typing `resume`.

Half the refreshed tokens went to re-reading the prior session.

05
CHAT-AS-CONVO

Chatting, not instructing.

Every message reloaded the full thread. Slow and expensive.

06
BLIND TRUST

Believing the docs.

Hallucinated API errors sent us chasing ghosts.

ACT 01 · OUR STORY · 03 OF 04

What we figured out.

Four realisations. Everything in this deck rests on these.

01

Plan before you prompt.

Brainstorm design with AI *before* asking for code. A clean spec is half the implementation.

→

02

Structure your side, not just your ask.

Codify the rules once — in a file the AI reads every session. Stop re-explaining the stack.

→

03

Fresh context, every new task.

New problem, new chat. Stop dragging dead context across unrelated work.

→

04

Token discipline is a skill.

Cap responses. Summarise before compaction. Never type `resume` again.

THE THESIS

AI is only as good as the structure you give it. The decision-making is still yours.

EVERYTHING AFTER THIS SLIDE IS PROOF.

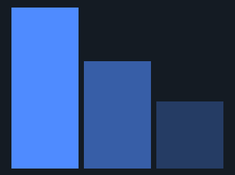
ACT 02 · THE FRAMEWORK · 01 OF 03

The 3 pillars.

Universal mental model. Any stack. Any team.

PILLAR 01

Context



Tell AI what your project is — once.

PILLAR 02

Patterns



Teach AI your rules — so it stops guessing.

PILLAR 03

Memory

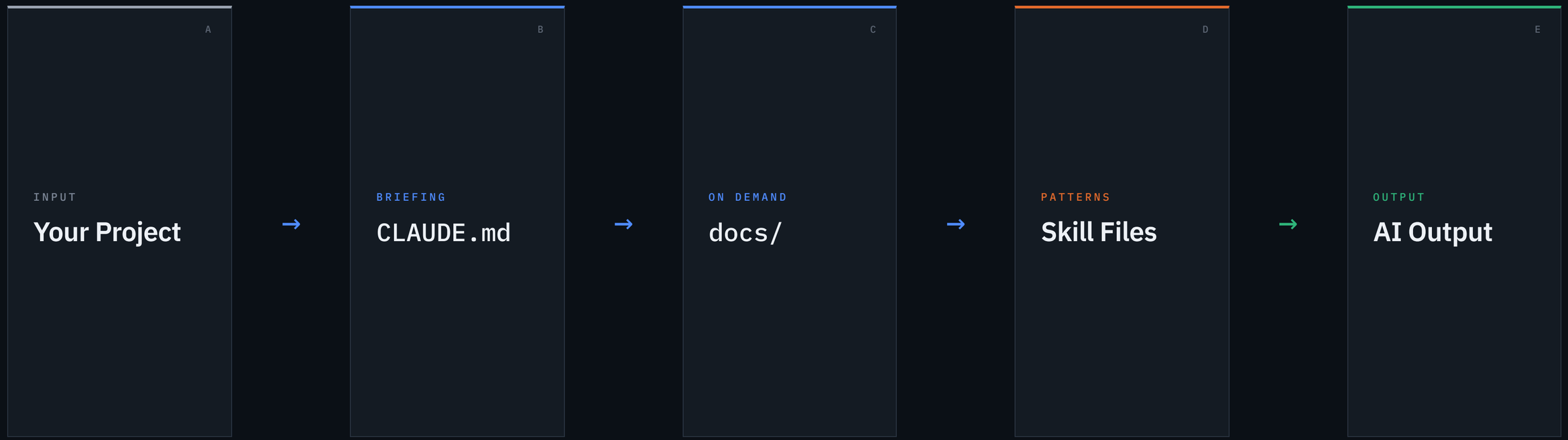


Decisions that survive session resets.

ACT 02 · THE FRAMEWORK · 02 OF 03

How it connects.

Five pieces. One flow.



ONE LINE *Briefing in. Patterns applied. Predictable output out.*

ACT 02 · THE FRAMEWORK · 03 OF 03

The Day 1 kit.

Six files. Every project. From the first commit. Start with one — layer the rest on top.

01	PROJECT BRIEFING	**CLAUDE.md** — Read every session. Stack, conventions, mandatory rules — the briefing file.	EST. 30 MIN
02	REFERENCE	**docs/** — Most-referenced domain knowledge. Loaded on demand, not on boot.	EST. 1 HR
03	YOUR RULES	**Skill files.** — Pick the pattern you repeat most. Codify once. Applied every time.	EST. 1 HR
04	PERSISTS	**Memory.** — Log first key decisions. Whatever AI shouldn't re-learn.	EST. 15 MIN
05	SAFETY NET	**open-questions.md** — Unresolved decisions tracked, not guessed. Blockers visible, not buried.	EST. 15 MIN
06	DAILY USE	**First slash command.** — The task you do most. Save it. Use it tomorrow.	EST. 30 MIN

ROLE TRACK 01 / 04 UI · DESIGN PIPELINE

UI.

HTML + JSX out beats screenshots in. Design tokens don't survive a screenshot — Claude Design hands us code that does.

EVALUATED	DEFAULT **Claude Design** HTML + JSX	EVALUATED **Figma MCP** COMPONENT PLUMBING	EVALUATED **UX Pilot & co.** SCREENSHOT IN
ACCURACY	**~95%+** Code-first; tokens preserved end-to-end.	**60 – 70%** Solid structure, drifts on tokens & states.	**Variable** High visual fidelity, low semantic fidelity.
DESIGN TOKENS	**Native** Reads `globals.css` `@theme` / Tailwind config.	**Partial** Mapped, sometimes drifts on naming.	**None** Pixels in → invented values out.
ITERATIONS	**One-shot, then nudge** Output is editable code; tweak in place.	**2 – 3 rounds** Re-prompt for tokens, spacing scales.	**Many** Each iteration is a fresh re-interpretation.
INTEGRATES WITH	**/ commands directly** JSX drops into Claude Code; `CLAUDE.md` applies.	**IDE plugin** Code lands in a sandbox, not the repo style.	**Standalone exports** Outside the project's rules & patterns.

DEFAULT TO CLAUDE DESIGN FOR HI-FI WORK *Tokens, types, conventions survive the handoff. Pixels never do.*

ROLE TRACK 02 / 04 BACKEND

Backend.

THE PROBLEM

Every session re-explained the stack. Errors were inconsistent. Tests were always "later."

WHAT WE BUILT

01	**CLAUDE.md**	Stack + 9 coding rules, loaded every session.
02	**Error skill**	10 scenarios mapped to exact exception classes.
03	**Test skill**	Auto-generated spec file after every feature.
04	**Spec-first**	Design the module before any code is written.

WHAT CHANGED

0 Accidental 500s

100% Features ship with tests

→ Checklist execution, not exploration

ROLE TRACK 03 / 04 FRONTEND

Frontend.

THE PROBLEM

AI guessed colors. Invented spacing. Used the wrong Tailwind classes. Every session started fresh.

WHAT WE BUILT

01	**Design tokens**	Colors, fonts, spacing inlined in `CLAUDE.md`.
02	**Slash commands**	`/build-screen` · `/add-component` · `/resolve-oq`.
03	**Reference JSX**	Exact layout from design tool. Read before code.
04	**PROJECT_REF.md**	Whole codebase map in one paste.

WHAT CHANGED

90% Correct UI on first pass

40 → 1 Session-boot file reads

0 Invented CSS values

ROLE TRACK 04 / 04 QA

Project management.

Our worst hit. And our biggest win.

30+MIN
PER TEST SESSION JUST ON TICKETING WORK

FRICTION

Manual creation.

Open tool, navigate, fill every field by hand.

→

EVAPORATES

Context lost.

Half the detail is gone by the time it's written.

→

VAGUE

No structure.

Devs spend 20 min asking what was meant.

→

NOISE

No dedup.

Same bug reported three times. Backlog fills.

→

NO DONE

No criteria.

Ships with nothing to verify against.

EVERY ISSUE, EVERY DAY

Open tool → navigate → fill every field → forget context → write vague description → miss duplicates → ship with no acceptance criteria.

QA TRACK · 02 OF 03

What we built for project management.

Four pieces. One system.

BRIDGE

MCP server

AI reads + writes to your real ticketing tool.

RULES

Layered instructions

Classify. Ask one question. Assemble silently.

THROUGHPUT

Batch mode

Paste all findings. Tickets drafted in parallel.

GUARDRAIL

Human in control

Nothing ships until you type `CONFIRM ALL`.

RAW PASTE → STRUCTURED TICKET

1 **Paste anything** → 2 **Classify silently** → 3 **One question max** → 4 **Dedupe check** → 5 **CONFIRM ALL**

QA TRACK · 03 OF 03

What changed.

~~30+ min~~ → **<10 min**

PER FULL TEST SESSION

QUALITY

Duplicates caught **before** they reach the backlog.

CONSISTENCY

Every ticket ships with criteria, repro, priority — automatically.

ACT 04 · TAKE THIS AND RUN · 01 OF 03

Ticketing, on rails.

Talk to Odoo from the IDE. No one starts cold — context travels with the ticket.

20	~75%	40–70
TOOLS EXPOSED VIA ODOO MCP	TOKEN REDUCTION PER SPRINT OP	LINES PER HANDOFF BUNDLE

THE SYSTEM · 3 PIECES

BRIDGE

Odoo MCP.

20 tools wired via `.mcp.json`. Reads, writes, chatter, attachments, bulk ops — all from the IDE in natural language.

COMMAND

`/ticket-context`

Move stage → check git → pick audience → generate markdown → attach & handoff. Five steps, one command.

ROUTER

Audience-aware.

Skill router infers the next role from changed files + stage name. BE → FE, FE → QA, QA → Release. No guessing.

THE HANDOFFS · 40 – 70 LINES, ROLE-SPECIFIC MARKDOWN

BE → FE BUNDLE

Backend handoff.

Endpoints, request/response shape, error envelope, auth role required, sample payloads.

FE → QA BUNDLE

Frontend handoff.

Affected screens, edge cases, what to click, what should break, perf & a11y notes.

QA → RELEASE BUNDLE

QA handoff.

Pass/fail matrix, regressions checked, screenshots attached, known-risk callouts.

ACT 04 · TAKE THIS AND RUN · 02 OF 03

Token economy.

Every token you waste is context you can't use.

~~~35k~~ → ~9k

TOKENS / SPRINT OP · AFTER THE FIX

THE PROBLEM

Unconstrained sessions blew up to **~35k tokens** per sprint operation. Claude scanned all files on first message — **30%+** of the context window gone before a single line of code was written. Long CLI outputs, full ticket reads, verbose responses — all eating into usable context.

FOUR LEVERS — THREE COMPRESSING TODAY, ONE COMPOUNDING WITH SCALE

01
COMPRESS RESPONSES

Caveman Mode

Strips filler from Claude's replies, keeps only technical content. Security warnings and multi-step ops stay uncompressed. Most tokens aren't in the code — they're in the surrounding explanation.

02
FILTER CLI OUTPUT

RTK

Intercepts `git`, `npm`, and test-runner output before it hits Claude's context. `rtk git status` → only what Claude needs to act on. Visible drop on any session with `npm install` or test runners.

03
STRUCTURED CONTEXT

`CLAUDE.md` + `docs/`

Root file loads lean. On-demand docs load only when the task needs them. Per-ticket summaries use a unified `get_ticket_summary` with 60s / 10m TTL caches instead of fresh reads every time.

04 WIRED · UNDER-USED
MAP THE CODEBASE

Graphify MCP

`/graphify` indexes the repo into a knowledge graph — tree-sitter on code (local), Claude on docs & diagrams. A pre-Grep hook nudges Claude to query `graph.json` first — `get_neighbors`, `shortest_path` — instead of scanning files. *"How does `orders` relate to `menu`?"* → one call, no file reads. **Set up, not yet earning its keep** — payoff compounds as the codebase grows.

THE RESULT

39 calls → 10 calls per sprint op (~75% reduction) — from levers 01-03. Lever 04 is wired but waiting on scale: when the codebase is big enough that orientation hurts, Claude reads the graph — not the files.
~~~

ACT 04 · TAKE THIS AND RUN · 03 OF 03

What moves the needle.

NON-NEGOTIABLE
ROLE-SPECIFIC
HIGH VALUE
AS NEEDED

WHAT	WHO	PRIORITY	INCLUDES	WHY
`CLAUDE.md`	Everyone	Non-negotiable	Stack · conventions · response envelope · 9 mandatory rules — loaded every session.	Without it, every session starts at zero.
Docs library	Everyone	Non-negotiable	`business-rules.md` , `order-lifecycle.md` , `learning/*` — ground truth, on demand.	Stops AI re-reading 40 files every task.
Memory system	Everyone	High value	Decision log · “built vs placeholder” tags · `open-questions.md` .	Decisions survive resets. No re-explaining.
Skill files	BE / FE	High value	`error-handling` · `unit-tests` · `a11y-checklist` · `redux-slice` .	Patterns codified once. Consistent output.
Slash commands	Frontend	High value	`/build-screen` · `/component-from-token` · `/resolve-oq` .	Reusable workflows. Single-line triggers.
Spec-first workflow	Backend	High value	Module design · DTOs · error map — settled before any code is written.	Implementation becomes checklist execution.
QA ticketing (MCP)	QA	Role-specific	Odoo MCP server (20 tools) and the `/ticket-context` command — batch mode, audience-aware.	3× faster test sessions.
Token discipline	Everyone	Ongoing	Cache, batch, parallel calls, scope — and never `resume` .	Compounds every session.
Plugins (Caveman, RTK)	Backend	As needed	Caveman token cap · RTK toolkit · Graphify MCP — when a blocker hits.	Add when token cost is the blocker.

CLOSE

> **When your setup is structured, you stop ~~managing the AI~~ and start working with it.**

Thank you.

→ Questions?

Speed
Less briefing. More shipping.

Consistency
Same patterns. Every PR.

Continuity
Context survives. Anyone picks up.

ONE LINE REMINDER

"The template is the starting point.
Your discipline is what makes it work."