THE PROBLEM

Claude Without Structure Is a Liability

> You are already using Claude. But without structure, you are building on sand.

Re-explanation Overhead

Every new session — same context, same stack, same rules. Explained again. From scratch.

Inconsistent Output

Same task. Different session. Different result. No enforcement. No guarantee.

No Guardrails

Claude deletes a config file. Edits `.env`. Pushes to the wrong branch. You find out after.

The stack you're about to learn exists to solve exactly these three problems.

CLAUDE CUSTOMIZATION STACK

What We're Building On

TWO SURFACES

Claude Code

Terminal / IDE Agent

CLI · VS Code · JetBrains

COMPONENTS

- CLAUDE.md
- Skills
- Commands
- Hooks
- MCP
- Plugins
- Agents

Claude.ai

Web / Projects Agent

Browser · Projects · API

COMPONENTS

- Projects
- MCP
- System Prompt
- Skills — limited — org level
- Plugins — published only

THE STACK

1. Commands
2. Skills
3. MCP Servers
4. Agents / Subagents
5. Hooks
6. Plugins

Each component kills one failure mode — re-explanation · inconsistency · no guardrails

COMMANDS

Named Shortcuts for Repeated Prompts

Give a name to a prompt

Type it once.
Call it forever.

`/command-name`

WHAT Triggers a saved prompt or workflow · Manually invoked — always explicit, never automatic

WHY Repetitive prompts = wasted tokens + inconsistency · Commands eliminate copy-paste prompting

WHEN Predictable, recurring tasks where you want manual control

`/code-review` `/deploy` `/standup`

FILE PATHS

```
~/.claude/commands/my-command.md  ← personal, all projects
.claude/commands/my-command.md    ← project-level
```

Manual trigger only — Claude never auto-invokes commands

SKILLS

Procedural Knowledge for Specialized Tasks

LLMs are great generalists — Skills make them specialists

PPT format rules · Code review checklist · Deploy procedure

PROJECT STRUCTURE

```
.claude/
└── skills/
    └── ppt-maker/
        ├── SKILL.md      ← instructions
        ├── examples/     ← sample outputs
        └── scripts/      ← helper scripts
```

WHAT `SKILL.md` with frontmatter + procedural instructions · Body loads only when invoked — zero cost until needed

WHY Specialized tasks need consistent methodology · Keeps heavy instructions out of main context

HOW Two trigger modes:

`/ slash command` · auto-detect

Skills ⊃ Commands — every command is a skill, not vice versa

MCP SERVERS

Claude's Connection to the Outside World

WHAT

Universal connector

Exposes external tools Claude can call like functions

Any MCP-compatible client can use the same server



`.mcp.json` ← project-level config

`~/.claude/mcp.json` ← global config

Auto-triggered — LLM decides when to call which tool

WHY

Agents were blind to the outside world

MCP gives Claude hands in real systems

→ Read
→ Write
→ Act

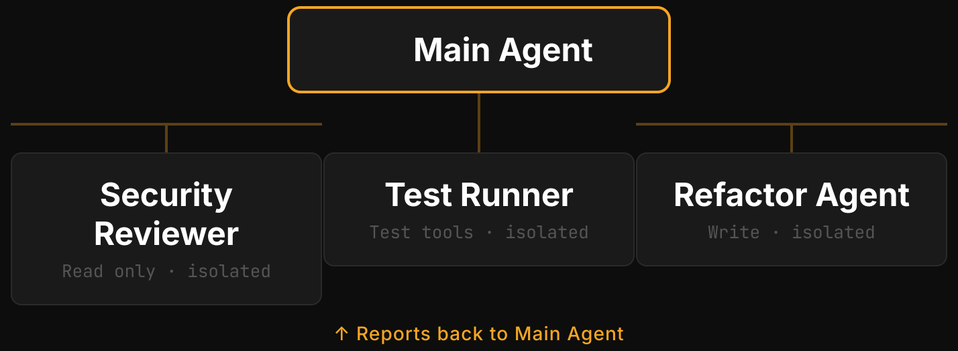
without leaving the IDE

AGENTS

Specialized Workers in Isolated Contexts

Context Window Overflow
One agent doing everything = bloated, degraded context

Lost in the Middle
Critical info buried deep in long context = missed by LLM



Context Isolation · Specialized Focus · Modularity · Parallelism

`.claude/agents/security-reviewer.md`

LLM decides · or explicitly requested

HOOKS

Guardrails the LLM Cannot Override

CLAUDE CODE — The Harness — obedient, faithful



Hooks are shell scripts — not prompts. The LLM has no say.

Your hooks live here

Protect sensitive files
`.env · config.json · never touched`

Block dangerous commands
`rm -rf · force push · pre-approved only`

Auto format & lint
Runs after every file write — silent, automatic

Telemetry & notifications
Log tool calls · Slack alerts · audit trail

`.claude/settings.json → "hooks": { "PreToolUse": [...] }`

PLUGINS

One Install. Full Stack.

The Starter Kit Box

One engineer builds the perfect stack.
Others shouldn't have to rebuild it piece by piece.
Bundle it. Publish it. Install it.

WITHOUT PLUGIN

Copy skills + configure hooks + add MCP + set up agents — manually, per developer

WITH PLUGIN

`/plugin install team-sdlc` — done

Plugin = distribution unit · MCP = protocol Not the same thing.

PLUGIN STRUCTURE

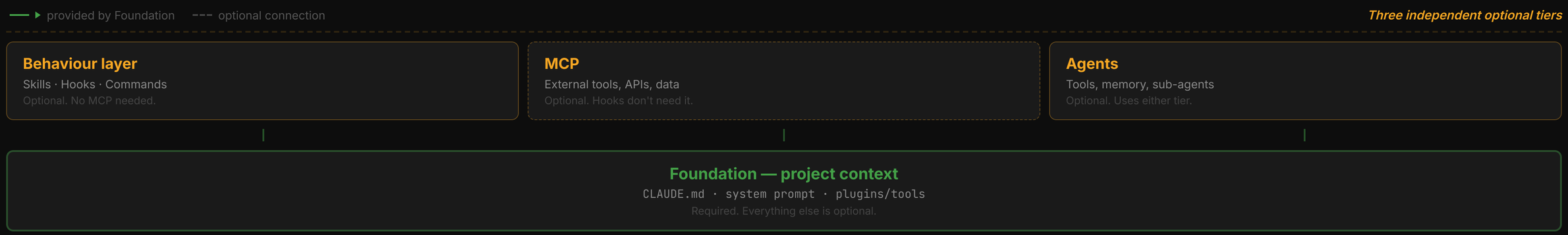
```
my-plugin/
├── .claude-plugin/
│   └── plugin.json   ← manifest
├── skills/
│   └── code-review/
│       └── SKILL.md
├── hooks/
│   └── prelint.sh
├── agents/
│   └── security.md
└── .mcp.json
```

Manual install — /plugin install <name>

COMPARISON

Right Tool. Right Job.

COMPONENT	LAYER	TRIGGERED BY	USE WHEN
`CLAUDE.md`	Foundation	Always — every session	Standing rules, conventions, always-on facts
`Commands`	Behaviour	You — manual `/name`	Predictable, recurring tasks you control
`Skills`	Behaviour	You or Claude — auto-detect	Specialized workflows, procedural knowledge
`MCP Servers`	Protocol	Claude — tool call	External system access across any MCP client
`Agents`	Orchestration	Claude or explicit	Isolated subtasks, context overflow risk
`Hooks`	Behaviour	Claude Code — lifecycle events	Enforcement, automation, safety — non-negotiable
`Plugins`	Distribution	Manual install	Team-wide setup, sharing a full configured stack



If it needs reasoning → Skill · If it needs enforcement → Hook · If it needs external access → MCP

SPEC-DRIVEN DEVELOPMENT

From Vibe Coding to Engineering Discipline

Vibe Coding

— Prompt → hope for the best
— AI guesses intent
— Inconsistent output
— Breaks at scale

Spec-Driven

+ Define before you build
+ AI works from contracts
+ Predictable, reviewable output
+ Scales across team & sessions

FEATURE SPEC TEMPLATE

Problem Statement	what & why
Functional Requirements	what it does
API Contracts	interfaces
Constraints	limits & rules
Edge Cases & Error Handling	failure modes
Acceptance Criteria	done means done

Skills + CLAUDE.md enforce the spec · Hooks protect it · Agents execute it

WHAT WE COVERED

The Stack. The Discipline. The Shift.

- Commands
- Skills
- MCP Servers
- Agents
- Hooks
- Plugins
- Spec-Driven Development

You can build all of these with Claude itself.

Vibe coding → Spec-driven development

From prompting an AI to engineering with one.