• CLAUDE CODE ARCHITECTURE GUIDE · UPDATED 2026

Matthias Herbert
(linkedin.com/in/matthiasherbert) Agentic Coder Trainer

◆ 2026 Edition · Claude Code

How to Design a CLAUDE.md That Actually Works

The architecture behind your AI agent's memory — scopes, frameworks, hooks, skills & the rules that separate chaos from clarity.

`~/.claude/CLAUDE.md` → `./CLAUDE.md` → `./CLAUDE.local.md` → `./src/CLAUDE.md`

Last scope wins on conflicts

01 THE 5 SCOPES (2026)

Global
`~/.claude/CLAUDE.md`
Personal defaults across all projects. Coding style, patterns, universal rules. Always loaded.

Project
`./CLAUDE.md`
Project-specific rules. Check into git & share with team. Build commands, conventions.

Local Secret NEW
`./CLAUDE.local.md`
Personal notes. Add to .gitignore — never shared with the team.

Folder
`./src/CLAUDE.md`
Module-level overrides. Scoped context for APIs, components. Loaded on-demand.

02 THE WHAT / WHY / HOW FRAMEWORK

WHAT
Give Context
- Project name & purpose
- Tech stack & versions
- Repository structure map
- Key dependencies
- Monorepo: what each app does

WHY
Set Principles
- Architecture decisions
- Code style & lint rules
- Naming conventions
- Anti-patterns to avoid
- Security constraints

HOW
Define Workflows
- `npm run build`
- `npm test` after every change
- `eslint . --fix` on save
- Branch & commit strategy
- Deploy & CI/CD steps

03 REAL-WORLD CLAUDE.MD & AGENTS.MD

You can write a CLAUDE.md by hand — or run `/init` in Claude Code. Neither gives you a solid production-ready starting point: `/init` only generates a bare-bones scaffold, while writing from scratch usually lacks consistent structure. We've put together a significantly better foundation — covering the WHAT/WHY/HOW framework, AGENTS.md for multi-agent workflows, hooks configuration and SKILL.md templates. Worth a look:

Agentic Coding Meta-Prompt
github.com/obviousworks/agentic-coding-meta-prompt

- CLAUDE.md template (≤200 lines)
- Hooks config (.claude/settings.json)
- AGENTS.md for multi-agent workflows
- SKILL.md templates & examples

04 BE SPECIFIC — VAGUE VS. PRECISE

Vague		Precise
✗ Write clean code	→	✓ Use camelCase for variables, PascalCase for React components
✗ Test everything	→	✓ `npm test` after every change, min 80% coverage for utils/
✗ Prefer TypeScript	→	✓ MUST use TypeScript strict mode. MUST NOT use `any` type
✗ Be careful with git	→	✓ Always create a new branch per task. NEVER commit to main directly

"CLAUDE.md is not a README for humans. It's ***onboarding docs for your AI teammate*** *— and every correction you add is a bug that never happens again."*
Boris Cherny · Creator of Claude Code · Staff Engineer @ Anthropic

Under **200** lines · ~ **150** instructions max · In **git** · Update **monthly**

05 7 RULES THAT MAKE IT WORK

1 Run /init first
Let Claude scaffold the baseline from your codebase, then curate. Don't start from scratch.

2 Stay under 200 lines
Too long = ignored. Claude attends to ~150 instructions reliably. Every line earns its place.

3 Hooks for 100% enforcement
CLAUDE.md is advisory (~70% followed). Hooks are deterministic — use for lint, test, security.

4 Use @imports for modularity
NEW Split configs via `@docs/git.md`. Cleaner, scoped, reusable across projects.

5 /compact & Plan Mode
NEW Run /compact at 50% context fill. Use Plan Mode for every task with 3+ steps.

6 Update it monthly
Living document. Every time Claude errs: add a rule. Boris calls it compound engineering.

7 Reference, don't duplicate
Point to `package.json` & tsconfig — don't copy. Use `@path/to/file` syntax.

+ SKILLS on-demand 2026
Domain knowledge in `.claude/skills/` — loaded when relevant, not every session.

06 ADVANCED PATTERNS — 2026 UPDATE

⚡ Hooks System
- Configure in `.claude/settings.json`, browse via /hooks
- **PreToolUse:** route risky ops to Opus for approval
- **PostToolUse:** auto-format code after every file edit
- **Stop hook:** verify work before Claude marks done
- Use /careful to block destructive commands

Skills System
- Place in `.claude/skills/SKILL.md` — on-demand only
- Invoke with `/skill-name` or applied automatically
- Embed `` !`command` `` for live shell output injection
- More transparent than MCPs — auditable, no black box
- Include libs so Claude composes, not rebuilds from scratch

Multi-Session (Boris's Method)
- Run 10–15 Claude sessions simultaneously in parallel
- Each session gets its own **git worktree**
- /compact at 50% context · /clear when switching tasks
- Plan → Execute → Verify: tests + linter + build always
- After every error: encode fix in `tasks/lessons.md`