EVOKNOW.AI · AI Super Power Skills for Claude

Real Engineering with Claude Code

A Claude Code user's guide to Matt Pocock's Skills for Real Engineers

Most AI coding failures are not model failures -- they are process failures. The agent builds the wrong thing, writes code with no feedback loop, or quietly rots your architecture. Matt Pocock's "Skills for Real Engineers" is a set of small, composable Claude Code skills that fix exactly those failure modes.

This guide is written for Claude Code users: install the plugin, set up a repo, and learn the handful of slash commands that turn Claude from an eager guesser into a disciplined engineer.

Repository: github.com/mattpocock/skills
Author: Matt Pocock · Published by EVOKNOW.AI
Guide version 1.0

1. What these skills are

"Skills for Real Engineers" is the set of agent skills Matt Pocock uses every day -- shipped straight from his own working directory. The pitch is deliberately narrow: real engineering, not vibe coding.

Frameworks like GSD, BMAD and Spec-Kit try to help by owning your whole process. The trade-off is that they take away control, and when the process itself has a bug it is painful to fix. These skills go the other way: each one is small, composable, and easy to adapt. They work with any model, and you are encouraged to hack on them and make them your own.

2. The four failure modes they fix

Each skill exists to close a specific gap that shows up when you code with an agent:

- Misalignment -- the agent did not actually understand what you wanted. Fixed by a grilling session: making the agent interview you before it builds.
- Verbosity -- the agent lacks your domain context, so it over-explains and under-delivers. Fixed by writing down a shared language for the project.
- Broken code -- the agent has no feedback loop. Fixed with tests and a disciplined debugging method.
- Poor architecture -- codebases degrade fast under an agent's hands. Fixed by investing in design on purpose rather than by accident.

The principle underneath all of it: "the rate of feedback is your speed limit. Never take on a task that's too big."

3. Installing in Claude Code

The skills are published as a Claude Code plugin, and it lives in Claude Code's official marketplace -- so there is nothing to add first, and updates arrive automatically. From your terminal:

```
claude plugins install mattpocock-skills
```

Or from inside a running Claude Code session:

```
/plugin install mattpocock-skills
```

That is the managed path: you subscribe to the set as a read-only bundle and get new skills as they ship. It is the right choice for most Claude Code users.

If you would rather own the files

There is a second path via skills.sh, which copies editable skill files into your project so you can hack on them. It works on any agent, Claude Code included:

```
npx skills@latest add mattpocock/skills
```

Nothing updates behind your back; you pull changes when you want them with "npx skills update". Pick one path or the other -- installing both leaves you with every skill twice. If you use this route, make sure setup-matt-pocock-skills is among the skills you select.

4. One-time setup per repository

Before using the engineering skills in a project, run this once in that repo:

```
/setup-matt-pocock-skills
```

It asks you three things and remembers the answers:

- Which issue tracker to use -- GitHub, Linear, or plain local files.
- Which labels you apply when triaging tickets (used by /triage).
- Where documents it creates should be saved.

That is the whole setup. You are ready to work.

5. Not sure which skill to use?

Start here -- it looks at your situation and routes you to the right skill:

```
/ask-matt
```

If you only remember one command from this guide, make it that one.

6. A typical workflow

The skills are composable, but they were designed to flow in a natural order. A realistic pass through a feature looks like this:

- /grill-with-docs -- the agent interviews you until the requirements are actually pinned down, sharpening your domain model and updating CONTEXT.md as it goes.
- /to-spec -- turn that conversation into a spec and publish it to your tracker. No second interview; it just synthesises what you already discussed.
- /to-tickets -- break the plan into tracer-bullet tickets, each declaring what blocks it, so the work can be done in safe order.
- /implement -- build the work, driving /tdd at the seams you agreed, and closing out with /code-review before anything is committed.

For work too large to hold in one session, /wayfinder plans it as a set of investigation tickets instead.

7. The engineering skills

You invoke these

- /ask-matt -- route to the right skill for your situation.
- /grill-with-docs -- grilling session that also builds the project domain model.
- /triage -- move issues through a triage state machine.

/improve-codebase-architecture -- scan for design opportunities and present them as a visual HTML report, then grill through the one you pick.

/to-spec, /to-tickets, /implement, /wayfinder -- the flow described above.

/setup-matt-pocock-skills -- the per-repo configuration step.

Claude reaches for these itself

/tdd -- red, green, refactor.

/diagnosing-bugs -- reproduce, minimise, hypothesise, instrument, fix.

/code-review -- dual-axis review against both standards and the spec.

/codebase-design -- design deep modules behind small interfaces.

/domain-modeling -- build and sharpen the project's shared language.

/research -- investigate against primary sources and save cited Markdown.

/prototype -- throwaway prototypes to answer design questions.

/resolving-merge-conflicts -- resolve conflicts by intent, not by hunk.

8. The productivity skills

/grill-me -- a relentless interview that keeps going until every decision branch is resolved. Use it for non-code work too.

/handoff -- compact a long conversation into a handoff document.

/teach -- multi-session skill teaching with a stateful workspace.

/writing-great-skills -- a reference for writing effective skills of your own.

/grilling -- the reusable interview loop that powers the grill commands.

9. Getting the most out of them

Grill before you build. It feels slow for two minutes and saves an afternoon.

Keep tasks small -- feedback rate is the speed limit. If a ticket cannot be verified quickly, it is too big.

Let CONTEXT.md grow. The shared vocabulary it captures is what stops the agent from guessing at your domain.

Do not install both distribution paths -- pick the plugin or skills.sh, not both.

Treat the skills as a starting point. They are small and hackable by design; if one does not match how your team works, edit it (via the skills.sh path) or write your own with /writing-great-skills.

Resources

Repository & docs: github.com/mattpocock/skills

Skill directory: skills.sh/mattpocock/skills

More skills, reviews, and guides: evoknow.ai/ai-skills