SuperDataScience

# SDS PODCAST

# EPISODE 1022: CLAUDE.MD, AGENTS.MD, SKILLS, HOOKS AND SUBAGENTS: A FIELD GUIDE TO STEERING AI AGENTS

| | | |
|---|---|---|
| Jon Krohn: | 00:00 | This is episode number 1022 on claude.md, agents.md skills, hooks, and subagents, all the info you need to steer AI agents effectively. Welcome back to the SuperDataScience Podcast. I'm your host, Jon Krone. Today's topic is the art of steering an AI agent, that is deciding where your instructions to an AI system should live so that they get followed reliably without bloating every request and racking up the bills. In episode number 1020, 1020 last week, I covered the two dials that control how capable an AI model is and how hard it works. Those are model size and effort level. Today's episode is the natural sequel because those LLM dials determine an agent's horsepower while today's topic determines the steering. If you've ever written careful instructions for an AI assistant, only to watch it ignore them an hour into a session, this episode explains why that happens and what to do about it. |
| | 01:03 | As with episode number 1020 last Friday, my jumping off point is a Claude Code blog post, this one written by Michael Segner, a member of Anthropic staff. And as usual, we've got a link to the full post for you in the show notes. And again, like last week, while I'm delighted to have Anthropic as such a big advertiser on the podcast this year, they have no influence on the topics I cover on the show. And later in the episode, I'll generalize today's guidance well beyond Claude, including to an open standard for agent instructions that OpenAI kicked off that Google backs and that has spread across the industry. So whichever tools you use, and even if you never write a line of code, actually, stick with me here through this episode. The Anthropic post that I was referring to a moment ago catalogs seven distinct methods for delivering instructions to ClaudeCode. |
| | 01:50 | And before your eyes glaze over at the number seven, here's the insight that makes them all snap into place. Every method differs along three properties, when the instruction loads into the model's context, whether it |

survives long sessions, and how much authority it carries. Master those three properties and you can reason about any instruction mechanism on any platform. A quick refresher connects this to last week in another way. As I covered in episode 1020, everything a model knows about your session, your messages, your files, its own prior reasoning lives in its context window. And every token in that window costs money and more subtly also costs your LLM's attention that's running behind your agent. When a session runs long, agentic tools perform what's called compaction. They summarize the conversation so far to free up room. Anything that was merely said earlier can get squeezed out in that summarization, which is why instructions you gave at the start of a marathon session mysteriously stopped being followed near the end.

02:52 The seven steering methods I'm going to cover in this episode are at heart, seven different answers to the question, how do I make this instruction cheap to carry and hard for my agent to lose track of? All right, let me walk through them now roughly from most familiar to most exotic. Method one is the claud.md file, a markdown file at the root of your project that loads at session start and stays in context for the entire session, getting reread after every compaction.That's helpful, means it's never going to disappear. This is where always relevant facts belong, build commands, directory layout, coding conventions, team norms, and so on. The catch is that every line costs tokens in every session, whether it's relevant or not. And in a shared repository, these files grow the way any unowned config file does. Every team appends its own instructions and nothing gets deleted.

03:45 Anthropic's advice is to keep this claw.md file under 200 lines, give it an owner, and review changes to it like you do code. There's also a clever variant. A claw.md file placed in a sub-directory loads only when the agent touches a file under that sub-directory. So team specific

conventions in a monorepo stay out of everyone else's context. All right, that was method one. Method two is rules. These are also markdown files like claw.MD files, but these express specific constraints like all API handlers must validate input with such and such a library. The superpower of rules is path scoping. A rule scope to your API directory stays out of context during a documentation only session and loads only when relevant files get touched. Method three is skills. Since Anthropic has been pushing these hard across their whole product line and several of my recent guests in recent months have been talking about how useful skills have been to them and I've been pressing them on how they use skills specifically.

04:49 So yeah, lots of Tuesday episodes with guests that have been digging into those. What are skills? Well, a skill is a folder containing a procedure, a deployment workflow, a release checklist, a code review playbook and so on. And here's the elegant part, only the skills name and one line description load at session start. The full body loads when the skill gets invoked either explicitly by you or because the agent matched it to the task. So procedures are the classic thing people wrongly stuff into their always on file. A 30 line deployment runbook doesn't need to be in context while you're writing documentation. So separate that into a skill file instead. So instead of having it be working globally, use skill files to be able to have specific context for specific kinds of activities that your agent might engage in. All right, method four is sub-agents, and this one deserves a moment because it inverts the whole problem.

05:47 The first three methods I was talking about are about getting instructions into context. Subagents are about keeping work, keeping context out of an agent. So a sub-agent is a defined assistant that runs a side task. It could be a deep search, a log analysis, a dependency audit, and it does it in its own fresh isolated context

window. So all those intermediate results that would clutter your main conversation never enter that main conversation. Only the sub-agent's final summary comes back to the main agent. These sub-agents can nest up to five levels deep and orchestrated workflows can coordinate tens to hundreds of background agents. When you hear multi-agent systems, this context isolation is a large part of what the fuss is about. All right, we are getting there. This is method five on hooks, and this one is actually the one we're going to spend the most time on.

06:42 Six and seven are really short. I only have a couple sentences on those, but this one I've got, I don't know, half a dozen sentences because I think it's really important. So hooks are the method I most want data scientists listening to internalize. Hooks should be deeply familiar to software engineers, their code, like a command, an HTTP call or a check that fire deterministically on lifecycle events like before a tool call or after a file edit. So things like run the linter after every edit, post a Slack on completion, inspect a command before it executes and block it, any of those kinds of things should be done with a hook. The anthropic blog post, the inspiration for this whole episode from, draws a distinction here that I'd frame as the biggest idea in that whole piece, which is that an instruction is a probability while a hook is a guarantee.

07:34 So if you write never do X in an instructions file for your agent, the model will comply most of the time, but under pressure or deep into a long session or when a prompt injection lurks in some file, it reads a prompted rule can fail. If something must never happen or something must always happen, the enforcement needs to be deterministic code like a hook, not a persuasive prose like a prompt. For those of us who spent years learning that data pipelines need validation checks rather than comments saying please don't pass nulls, this lesson will feel familiar. All right. And then yeah, method six and seven,

they're output styles for number six and appending the system prompt for number seven. Both of these methods modify the system prompt itself, which carries the highest instruction following weight of anything here. They're the blunt instruments. Output styles can replace the agent's entire default role and silently strip built in behaviors like running tests before declaring work complete.

08:33 So handle output styles with care. While appending to the system prompt on the other hand is additive and better suited to tone and formatting preferences. So that's it. Those are the seven claw.md file, rules, skills, sub-agents, hooks, output styles, and appending the system prompt. That's the claud code taxonomy for getting your agents to behave the way that you want them to. Now let's generalize because the ideas here have escaped Anthropic's walls in a big way. The clearest example is a file called agents.md. OpenAI released this convention in 2025 as an open plain markdown standard for the same job as a root claude.md file, a file in your repository that tells any AI coding agent how to build, test, and contribute to the project. Stewardship for agents.md files then moved to the Agentic AI Foundation under the Linux Foundation, the same body that stewards Anthropics model context protocol, MCP, and with backing from Google, Microsoft, AWS and others.

09:37 So agents.md files are a broad standard now. And at the time of recording, agents.md is read natively by OpenAI's Codex, by Cursor, by GitHub Copilot's coding agent, by Google's Gemini CLI, sorry, by Windsurf and by dozens of other tools. And it has been adopted by upwards of 60,000 repositories with monorepos nesting one per package the same way ClaudeCode nests subdirectory files. If your team uses multiple coding agents, the emerging best practice is to make agents.md your single source of truth and have each tool's native file point at it. And here's a research finding on this that I found interesting. A new academic study by researchers at

Ateha Zurich in Switzerland examined across 138 real world repositories to find that developer written instruction files improved agent task success rates modestly about 4%, but it cut agent introduced bugs by 35 to 55%. That is mega. That same study found that instruction files generated by an LLM rather than written by developers who know the code base decreased success rates while increasing inference costs by over 20%.

10:52 So let me reiterate the importance of this. If the instruction files are written by a human developer, they saw task success rates increase modestly. If these instructions are written by an LLM, they see success rates actually decrease. And if the instructions are written by a human developer, we see agent introduced bugs decreased by up to 55% while when we do it with an LLM, we see inference costs increase by over 20%. So in other words, these data suggest the value is in the judgment, the human judgment encoded in the file, not the file's plain existence. You can't delegate the steering wheel to the thing being steered. Beyond that study, which is interesting and the agents.md file, other methods are converging across vendors too. OpenAI's codex has adopted skills for reusable procedures, deprecating its earlier custom prompts feature in their favor, has hooks for enterprise audit trails and lifecycle automation and has made subagents generally available with support for a handful of concurrent agents.

11:57 Skills themselves, originally an anthropic format, have become an open standard adopted by dozens of tools these days. Whichever agent harness you build on, the taxonomy transfers. And for listeners who don't write code, you've been steering AI systems with a simplified version of this framework all along. ChatGPT's custom instructions are your always on root file, global loaded into everything best kept short. While ChatGPT's projects, Claude's projects, and Gemini's gems are your path scoped rules, instructions and files that load only within

a defined workspace so your marketing voice guide doesn't leak into your tax questions, for example. Custom GPTs and gems from Google configured for a single task are the consumer cousin of Claude's skills. The same principles apply at every level of sophistication. Global instructions should be few and short and everything else should be scoped to load only where it's relevant. All right, so let me wrap up the meat of today's episode with three takeaways you can apply this week, whatever your stack.

13:00 So first, match persistence to relevance. Facts the agent should hold at all times go in the always on file, kept ruthlessly short. Procedures and area specific conventions go in scoped mechanisms that load on demand. If your instructions file has a step-by-step process in it, that process wants to be a skill. Second of three takeaways, know the difference between an instruction and a guardrail. Always and never, those words are signals that you've left the land of prompting and need deterministic enforcement hooks, permissions, or their equivalents because a model following an instruction is a probability and probabilities fail at scale. My third and final takeaway is to treat steering files as code. Give them a human owner, review changes and prune those files because an instructions file that grows without gardening dilutes adherence to the instructions that matter. And per the Eteha Zurich study I mentioned, handcrafted beats auto-generated by a wide margin here.

14:02 Between last Friday's episode now and today's, you have the full control panel, model size for capability, effort for thoroughness, and a well-organized instruction hierarchy for direction. That combination, knowing not only how to make these systems powerful, but how to make them precisely reliably yours is fast becoming a defining skill of our field. Go forth, listener, and steer wisely. Your agents await their commands. All right. And then as we do when I remember to do it, we're going to wrap up with an Apple

podcast review. This one is from someone with a name that's like Secure Girl or Sekur Girl, S-E-K-U-R-G-R-L. And the title of our podcast review is Best AI Podcast. She says, I assume it's a she here, "I have been listening to this podcast for over a year and it is one of my favorite podcasts, informative guests with real life expertise and varied backgrounds, leading edge developments, and the host approaches topics both with curiosity, knowledge and academic objectivity I would recommend to anyone who wants to learn more about AI and who doesn't." Well, thank you Secure Girl or Secret Girl.

15:14 Really appreciate the Apple Podcast feedback and thanks to all you listeners for recent ratings and feedback on Apple Podcasts, Spotify, and all the other podcasting platforms out there as well as for likes and comments on our YouTube videos. Bonus points if you leave written feedback in Apple Podcasts. I think that is one of the most helpful things you can be doing to get the word out about this show and make sure that we continue to be going forever. So if you do write Apple Podcast reviews, I will read them on air like I did today. At this time, I'm mostly only seeing those done in the US Apple Podcast platform, but at some point I am going to, when I have a little bit more time, I'm going to spend some time looking through other countries' Apple Podcasts reviews because I know that we do have some from other countries as well and I'll get to that backlog eventually.

16:01 So thank you very much and that is the end of today's episode. Yeah, I hope you enjoyed it. I hope it was helpful for you and I hope that you'll keep coming back and keep on listening. Until next time, keep on rocking it out there and I'm looking forward to enjoying another round of the SuperDataScience Podcast with you very soon.