

CODING AGENTS

A practical intro for engineers new to coding agents

Start simple. Scale smart.

Dan Vega

Spring Developer Advocate · Broadcom



slides and code

HELLO 🖐️

About me.

Spring Developer Advocate at Broadcom

Java Champion

Co-author of Fundamentals of Software Engineering

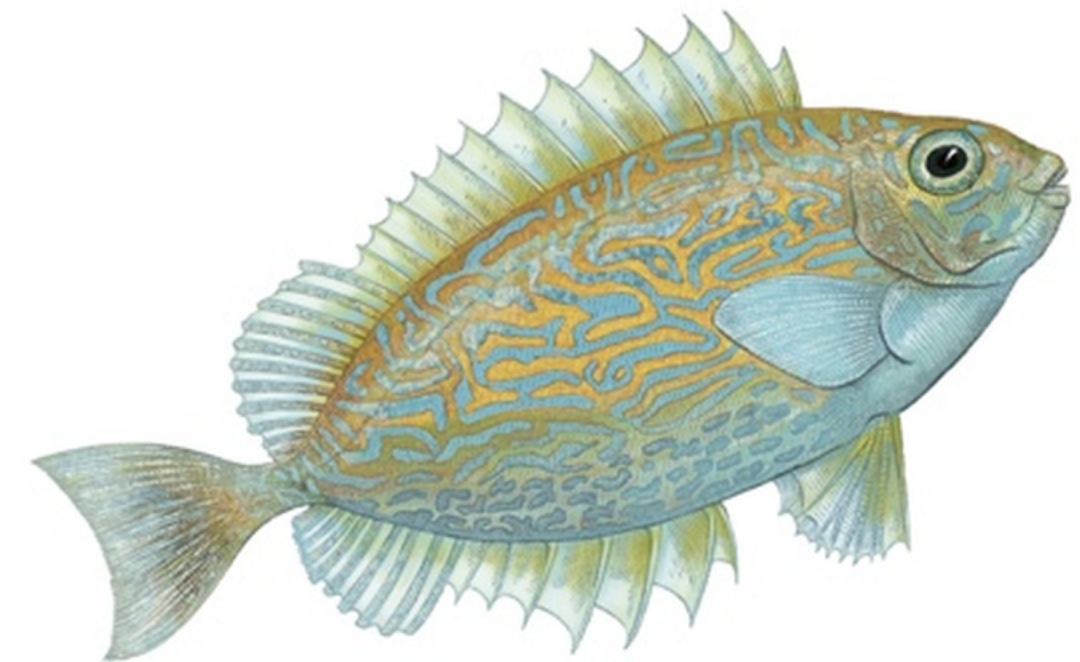
25 years as a software engineer

danvega.dev

O'REILLY®

Fundamentals of Software Engineering

From Coder to Engineer



Nathaniel Schutta
& Dan Vega

One quick disclaimer.

**If you're already running five terminals
with custom agents and skills...**

This talk probably isn't for you. Go grab another coffee. We'll be in the fundamentals for most of this talk.

CODING AGENTS

One tool today. Most of it translates to **your
favorite.**

Codex Gemini CLI Copilot Cursor Windsurf Grok Build Kiro

**I want to describe a feeling I get
these days.**



Morgan Lee

@shipwright_dev · 2m 

five agents in five tabs, numbered. a notification pings me when one needs an answer. I barely touch the keyboard now





Kay Greenfield

@greenfield_kay · 7m 

**reviewed three PRs from the gym this morning. my
agents opened four more while I was in the shower**





Four Terminals

@fourterminals · 11m · 

**if you're not orchestrating subagents in 2026,
you're already obsolete**





Cronjob Carla

@cronjob_carla · 18m · 🌐

**set up a routine that triages our backlog at 6am.
it closed two issues before standup and
nobody noticed for a week**





NullPointer Nate

@nullpointer_nate · 24m · 

**20 years in this industry and I have never
felt this far behind**



18



42



97



23

**Social media driven
development.**

QUICK SHOW OF HANDS...

Who's feeling overwhelmed?

It's okay. This is a safe space.

Things we're supposed to learn in **coding agents**.

CLAUDE.md

Hooks

Subagents

MCP

Worktrees

Plan Mode

Skills

Agents

/clear

SKILL.md

Memory

/init

Plugins

Agent Teams

Things we're supposed to learn in **coding agents**.

CLAUDE.md

Hooks

Subagents

MCP

Worktrees

Plan Mode

Skills

Agents

/clear

SKILL.md

Memory

/init

Plugins

Agent Teams

Harness

Checkpoints

PostToolUse

Status Line

/context

Sonnet

Permissions

Routines

Tokens

/goal

Compaction

PreToolUse

Marketplace

/resume

Things we're supposed to learn in **coding agents**.

CLAUDE.md

Hooks

Subagents

MCP

Worktrees

Plan Mode

Skills

Agents

/clear

SKILL.md

Memory

/init

Plugins

Agent Teams

Harness

Checkpoints

PostToolUse

Status Line

/context

Sonnet

Permissions

Routines

Tokens

/goal

Compaction

PreToolUse

Marketplace

/resume

SessionStart

Auto Mode

Context Window

Side Chat

Background Sessions

Slash Commands

Headless Mode

Opus

Dispatch

Auto-Accept Edits

Cowork

Effort Levels

Haiku

.claude/

Computer Use

...and that's just this week

You are not behind.
You don't need to learn all of this.
You need the building blocks.

Scale later. We're all on our own journey.

TODAY

What we'll cover.

01 The fundamentals

agent · harness · token · context window

02 Getting started

three surfaces · permissions · plan mode

03 Context is the job

CLAUDE.md · skills · MCP

04 Scaling up

subagents · hooks · goals · worktrees

SECTION 01 / 04

The fundamentals.

The vocabulary everything else hangs on.

01 **The fundamentals**

02 Getting started

03 Context is the job

04 Scaling up

FUNDAMENTALS · 01

Agent.

FUNDAMENTALS · 02

Harness.

HARNES

The machinery that builds and runs the loop.

Claude Code is a harness.

So are Codex, Cursor, Gemini CLI, and Copilot's agent mode.

Same model, different harness, **different results.**

harness · the car

model · the driver

you · the destination



decide



use a tool



observe

↺ repeats until the goal is done

tools

steering and pedals
read, edit, run

context

windshield and mirrors
what the model sees

safety rails

brakes and seatbelts
sandbox, approvals

orchestration

cruise control
retries, loops

Agent = model + harness. You pick the destination.

FUNDAMENTALS · 03

Token.

TOKEN

You don't pay for words. You pay for **tokens**.

Ref

actor

the

payment

service

and

run

the

tests

= 9

Roughly three quarters of a word. Claude reads, writes, and bills in them.

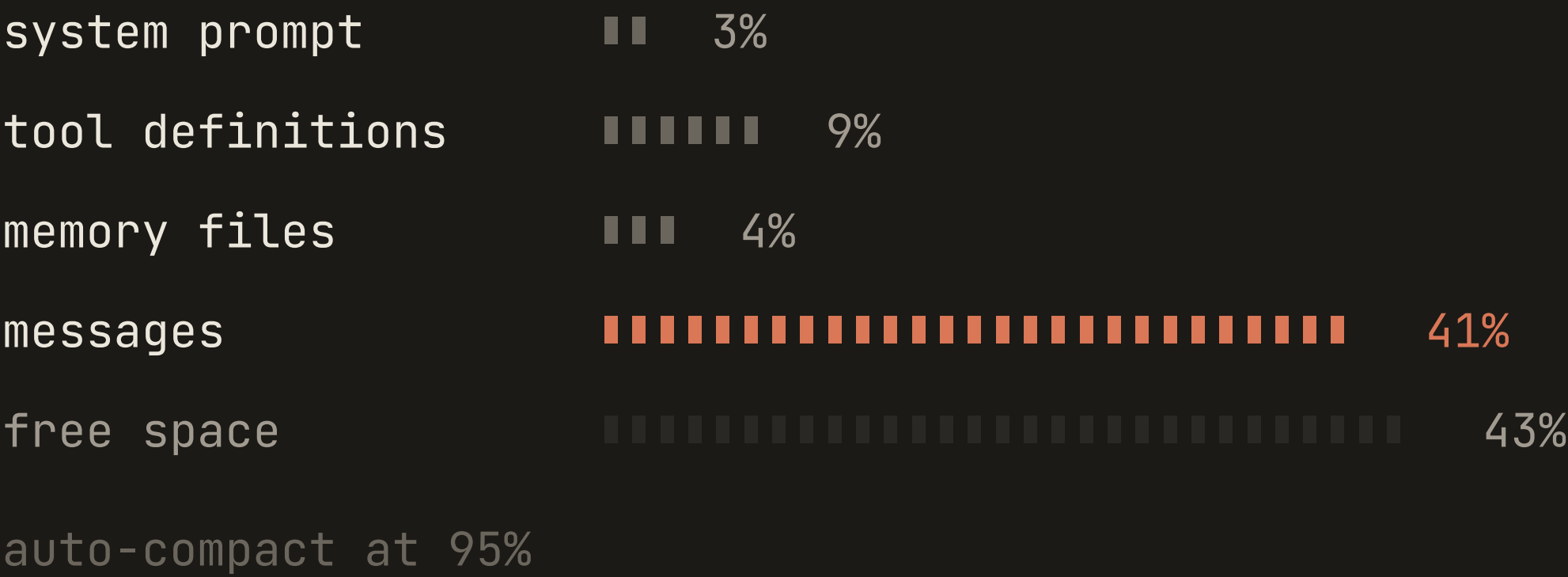
FUNDAMENTALS · 04

Context window.

CONTEXT WINDOW

Working memory for one session.

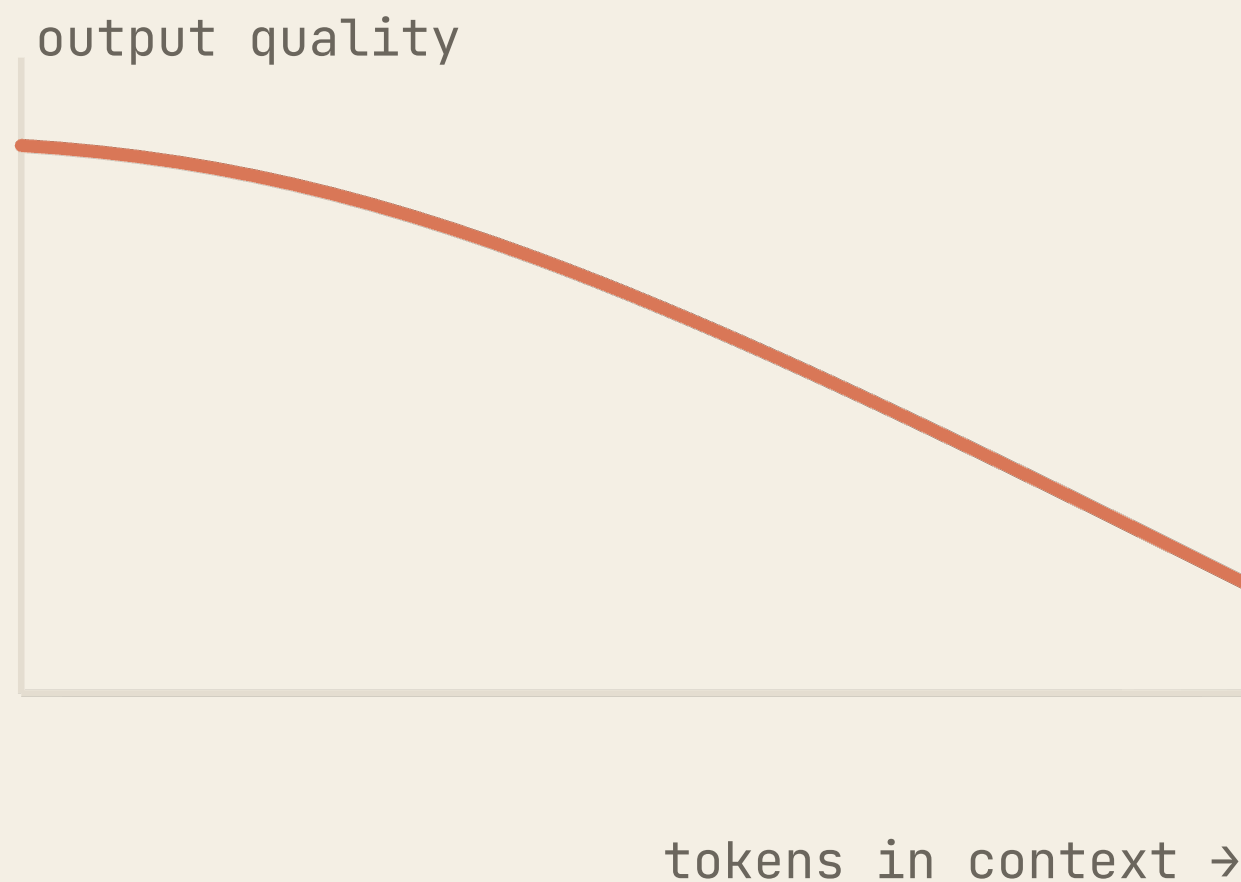
> /context



Everything Claude is holding in its head right now. Big, but finite.

THE CATCH

Context rot.



More context is not better context.

18 frontier models tested. All 18 got worse as the window filled.

Source: Chroma, "Context Rot" (2025)

SECTION 02 / 04

Getting started.

Three surfaces, permissions, plan mode.

01 The fundamentals

02 Getting started

03 Context is the job

04 Scaling up

THREE WAYS IN

Same engine underneath.

terminal

fast, scriptable

desktop

parallel sessions

web

runs while you're away

BEFORE YOU START

Two ways to pay.

\$/month

Subscription. Flat, predictable.

Pro, Max, or a Team seat. One pool across chat and Code.

\$/token

API. Metered, no ceiling.

Great for scripts and CI. Pricey for daily coding.

A stray `ANTHROPIC_API_KEY` silently moves you to the right column. `/status` shows your route.

DEMO · 01

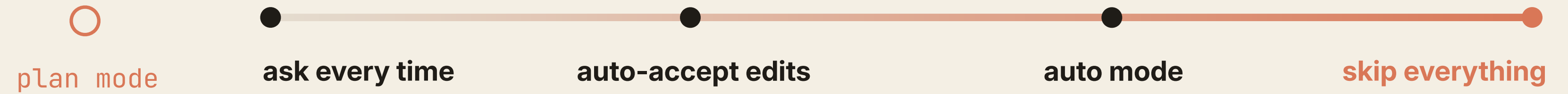
A first session.

Open a project. Run claude. Ask for one small thing. Watch the loop.

→ `jump to claude code`

TRUST, BUT VERIFY

Permissions.



THE ONE FOLDER TO KNOW

.claude/

Everything custom about your Claude
Code setup lives here. Memory.
Commands. Agents. Skills. Hooks.

```
.claude/  
├─ commands/your custom slash commands  
├─ agents/subagent configs (markdown)  
├─ skills/SKILL.md folders  
├─ hooks/lifecycle shell scripts  
├─ settings.json project settings  
└─ settings.local.json your overrides (gitignored)  
CLAUDE.md project memory (root of repo)
```

INSIDE .CLAUDE

settings.json

One JSON file controls permissions, environment, and team-wide defaults. Checked into the repo, it travels with the project.

.claude/settings.json

```
{
  "model": "sonnet",
  "permissions": {
    "allow": ["Bash(./mvnw *)", "Bash(git diff *)",
      "Read"],
    "deny": ["Bash(rm -rf *)", "Bash(mvn deploy *)",
      "Read(./env)"],
    "ask": ["Bash(git commit *)", "Bash(git push *)"]
  },
  "env": {
    "SPRING_PROFILES_ACTIVE": "test"
  }
}
```

MAYBE MY FAVORITE FEATURE

Plan mode.

Hit  Tab

until plan mode. Claude goes read-only.

The difference between "oh no" and "yes, do exactly that."

- 01 Explore the codebase
- 02 Ask clarifying questions
- 03 Write a markdown plan
- 04 You approve · it executes

NO PLAN MODE?

Prompt it. Or make it a skill.

PROMPT IT

Ask for the plan first.

"Explore the code. Ask me questions. Write a plan.
Don't edit anything until I approve it."

SKILL IT

Write it down once.

Save that prompt as a skill. Any agent that
supports skills can use it.

More on skills in a bit.

DEMO · 02

Permissions and plan mode.

A new project, from first prompt to trusted setup.

→ [jump to claude code](#)

SECTION 03 / 04

Context is the job.

Steering Claude with what you already know.

01 The fundamentals

02 Getting started

03 Context is the job

04 Scaling up

This is where you can take a coding agent from a nice parlor trick to an irreplaceable teammate.

The model is only as good as the
context you give it.

THREE GAPS, THREE FIXES

Three ways the model comes up short.

GAP 01

CLAUDE.md

What it should always
know about this project.

GAP 02

Skills

What it should know how to
do.

GAP 03

MCP

What it cannot see from
where it sits.

PROJECT MEMORY

CLAUDE.md

A markdown file at the root of your repo.
Claude reads it every session.

20 minutes on a good one makes every session sharper.

CLAUDE.md

```
# spring-petclinic
Java 25 · Spring Boot 4.1 · PostgreSQL · Maven

## Conventions

- Constructor injection only
- Records for DTOs
- Tests use Testcontainers, not H2

## Run it

$ ./mvnw spring-boot:run

## Gotchas

- Migrations live in db/migration, not resources/
```

Have an **AGENTS.md**? Claude reads it when there's no **CLAUDE.md**.

PROJECT MEMORY · THE CRAFT

Write one that actually helps.

VAGUE · DON'T

"Write good code."

"Follow best practices."

"Format things properly."

"Test your changes."

Claude already tries to do all of this. Zero signal.

SPECIFIC · DO

"Use 2-space indent, no semicolons."

"Constructor injection only."

"DTOs are records in model/."

"Run ./mvnw verify before done."

Concrete rules Claude can't guess. Real signal.

Structure it.

Headings and bullets, not prose.

Keep it tight.

Every line costs context.

Revisit it.

Fix the file, not just the chat.

KEEP IT LEAN

Progressive disclosure.

Point at the details instead of pasting them. Claude reads a file only when the task needs it.

CLAUDE.md

```
# Payments API  
Constructor injection only.  
Run ./mvnw verify before done.  
Testing conventions: @docs/testing.md  
DB migrations: @docs/migrations.md  
Release process: @docs/release.md
```

Loaded up front: 8 lines. Loaded on demand: everything else.

GAP 02 · WHAT IT SHOULD KNOW HOW TO DO

A skill is a folder with a SKILL.md.

Expertise packaged once, pulled in on demand. Not crammed into every prompt.

SKILLS

What one looks like.

The description is the trigger. Claude reads it to decide when the skill applies.

The body is the playbook. Steps, conventions, commands.

It's just markdown. In the repo, reviewed and versioned like code.

.claude/skills/readme/SKILL.md

name: readme

description: Write, rewrite, or review a README so it serves its actual reader. Use whenever the task is creating or improving a README.

A README has one job: get the reader to what they came for in under a minute.

Ground rules

- Open with one plain sentence saying what the project is and who it's for
- Verify every command against the repo
- No hype. Show a code sample instead.

SKILLS

If you explain it twice, make it a skill.

`new-endpoint`

scaffold a REST endpoint the way this team does it

`db-migration`

write a Flyway migration, naming and all

`code-review`

review a diff against the team's checklist

`release-notes`

turn merged PRs into a changelog, in house style

SKILLS IN THE WILD

Four jobs, twelve skills.

VIDEO

video-edit

Raw footage to Premiere.

video-rough-cut

Cut dead air, fillers, and retakes.

video-motion-graphics

Graphics timed to what I say.

WRITING

blog-new-post

A first draft in my voice.

blog-seo-optimize

Find what developers search for.

newsletter-publish

Beehiiv edition to my site.

SHIP IT

shipit-shape

Find the smallest useful version.

shipit-mvp

PRODUCT.md before any code.

shipit-feature

Build the next feature and verify it.

PODCAST

spring-office-hours-edit

Raw stream to podcast mp3.

spring-office-hours-prep

Audio and show notes in one go.

podcast-appearance

Log my guest appearances.

Type the same prompt twice? Make it a skill.

GAP 03 · WHAT IT CANNOT SEE FROM WHERE IT SITS

MCP is a standard plug for outside tools.

One protocol. Any tool that speaks it, Claude can see and use.

M C P

What connecting looks like.

One command to connect. The server does the rest.

Claude discovers the tools. The server advertises what it can do.

You stay in charge. Tool use runs through the same permissions.

terminal

```
$ claude mcp add --transport http \
    dvaas https://mcp.danvega.dev/mcp

✓ Connected to dvaas
✓ 21 tools discovered:
youtube · blog · speaking ·
newsletter · podcasts
```

M C P

Wherever Claude is guessing, plug in the source.

github

read the real issue, comment on the real PR

postgres

query the actual schema, not a guess at it

chrome-devtools

drive a real browser: live DOM, console, network

figma

build from the design file, not a screenshot

cloudflare

manage workers, DNS, and logs on your real account

context7

current docs for your framework version, not training data

CONTEXT IS THE JOB

Skills versus MCP.

SKILLS

Teach Claude how.

Markdown playbooks for repeatable work. Static.
Can't reach outside the repo.

```
"turn a newsletter into a site post"
```

MCP

Give Claude tools.

Live connections to tickets, databases, APIs.
Capabilities. They don't teach taste.

```
"fetch the edition from Beehiiv"
```

TOGETHER MCP fetches the newsletter. The skill publishes it my way.

DEMO · 03

Skills and MCP.

Teach it a skill, then plug in `mcp.danvega.dev`.

→ [jump to claude code](#)

SECTION 04 / 04

Scaling up.

One session to many. Watching to delegating.

01 The fundamentals

02 Getting started

03 Context is the job

04 Scaling up

SCALING UP

An agent is a file.

.claude/agents/code-reviewer.md

name: code-reviewer

description: Reviews code for quality

tools: Read, Glob, Grep

model: haiku

You are a code reviewer. Give specific,
actionable feedback.

Its own prompt

you decide how it behaves

Only these tools

read only, in this case

A cheaper model

haiku for mechanical work

Its own context window

the whole reason to use one

It reads 40 files. You get back a paragraph.

DELEGATING SAFELY

Hooks. Hard rules, not vibes.

CLAUDE.md

Advises.

The model reads it. The model can drift.

"Use `BigDecimal` for money."

Claude usually listens.

H O O K S

Enforce.

A script that runs before or after Claude acts.

Every time, no judgment call.

Block any edit that stores money in a double.

Claude can't get past it.

The more you delegate, the more you want rules that cannot be talked out of.

WORK THAT REPEATS

Routines.

Claude works while your laptop is closed.

	WHEN		CLAUDE
schedule	Every weekday, 8am	→	Triage new issues and label them.
api	CI goes red	→	Read the log and open a fix PR.
webhook	A PR opens	→	Review it and leave comments.

FROM WATCHING TO DELEGATING

/goal

Set the outcome, not the task. Claude keeps working, turn after turn, until it gets there.

```
/goal Every task in PLAN.md is marked complete,  
and ./mvnw verify passes with output shown for each one.  
A task is not complete until it has a unit test  
and an integration test covering its behavior.  
Update PLAN.md after each task.  
Stop after 25 turns.
```

← THE OUTCOME

← HOW IT'S CHECKED

← WHEN TO STOP

KNOW THE DIFFERENCE

/goal is not spec driven development.

SPEC DRIVEN

A plan.

Every decision made up front. The agent builds to it.

/GOAL

A finish line.

Keep going until this is true. It doesn't decide what to build.

Don't reach for it when the task needs design decisions you haven't made yet.

MY TAKE

You may not need spec-driven development.

VIBE CODING

One prompt to rule them all.

Fun for prototypes. Without a plan, it falls apart in production.

SPEC DRIVEN

A spec for everything.

Spec Kit, Kiro, OpenSpec, BMad. Great for big teams. Overkill for one developer.

My middle ground: one PRODUCT.md and my own process, written as skills.

RUNNING MORE THAN ONE

Two agents, one repo, **one mess.**

Git worktrees give each branch its own folder. Same history, separate working files.

FROM CLAUDE

```
claude --worktree feature-auth
```

OR BY HAND

```
git worktree add ../api-auth auth-fix
```

```
git worktree remove ../api-auth
```

One .git history, shared across all of them.

Separate files, so builds and tests don't collide.

No stashing, no branch switching mid task.

Cost: each folder needs its own install and build.

DEMO · 04

Agents.

Working with subagents

→ [jump to claude code](#)

FROM PAIR TO PARALLEL

Gates, not babysitting.

You can't watch five agents. Your gates can.

	WHEN		WHAT IT CHECKS
permissions	Before it acts	→	Allow the safe stuff. Deny the rest.
hooks	While it works	→	Hard rules it can't talk its way past.
./mvnw verify	Before it says done	→	Tests pass, or it isn't done.
ArchUnit	In every build	→	Architecture rules it can't quietly break.
code-reviewer	Before you merge	→	A second agent reads the diff.

Remember the wall? **You know the ones that matter.**

CLAUDE.md

Hooks

Subagents

MCP

Worktrees

Plan Mode

Skills

Agents

/clear

SKILL.md

Memory

/init

Plugins

Agent Teams

Harness

Checkpoints

PostToolUse

Status Line

/context

Sonnet

Permissions

Routines

Tokens

/goal

Compaction

PreToolUse

Marketplace

/resume

SessionStart

Auto Mode

Context Window

Side Chat

Background Sessions

Slash Commands

Headless Mode

Opus

Dispatch

Auto-Accept Edits

Cowork

Effort Levels

Haiku

.claude/

Computer Use

You are not behind. Start simple, scale smart. **We're all on our own journey.**

SAVE YOURSELF A FEW WEEKS

Mistakes I made so you don't have to.

- 01 **Vague one-line prompts.** "Fix the bug." Cool. Which one? Where? What did you try? Be specific.
- 02 **Skipping plan mode for real work.** Five minutes of planning saves an hour of "what did you just do?"
- 03 **Never running /clear.** Stale context bleeds into new tasks. Wipe between unrelated jobs.
- 04 **No CLAUDE.md.** Re-explaining your stack every session. Twenty minutes, write it once.
- 05 **Trusting the first answer.** Read the diff. Run the tests. Push back. Claude likes the pushback.
- 06 **Reaching for Skills/MCP/Agents too early.** First learn what Claude does well bare. Then add scaffolding.



github.com/danvega/claude-code-start-simple

That's it.

Thanks. Questions?

Dan Vega

Spring Developer Advocate · Broadcom

danvega.dev · [@therealDanVega](https://twitter.com/therealDanVega)