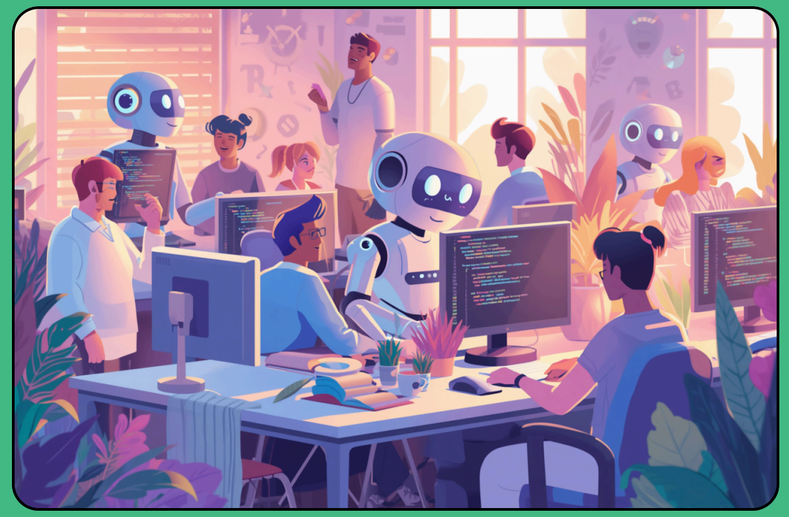
Being a Vue Developer in the Age of AI

Erik Hanchett · VueConf 2026

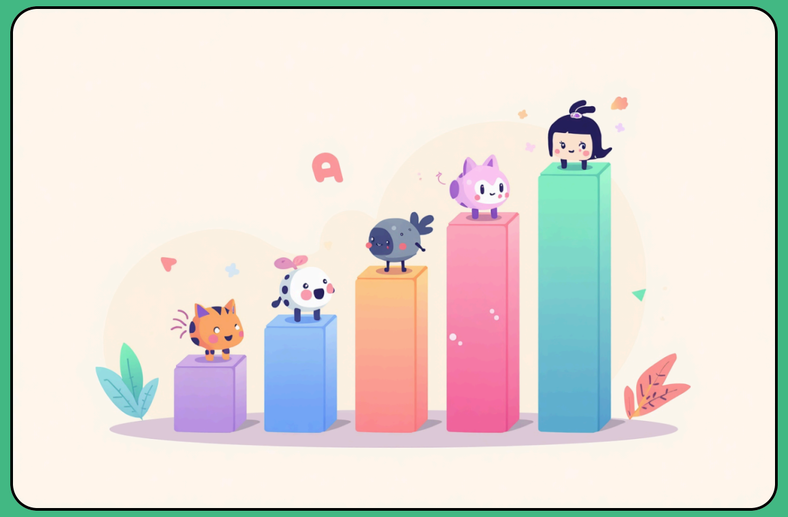
The Shift to AI Coding

The Shift Is Real: AI-Assisted Coding is Mainstream



Discovery

Devs are trying AI tools individually, no team standards yet.



Adoption

Teams use AI intentionally with rules and feedback loops.

Integration

AI is becoming a **core part** of coding workflows.

I haven’t written code by hand in over 6 months

From Code Writer to Conductor

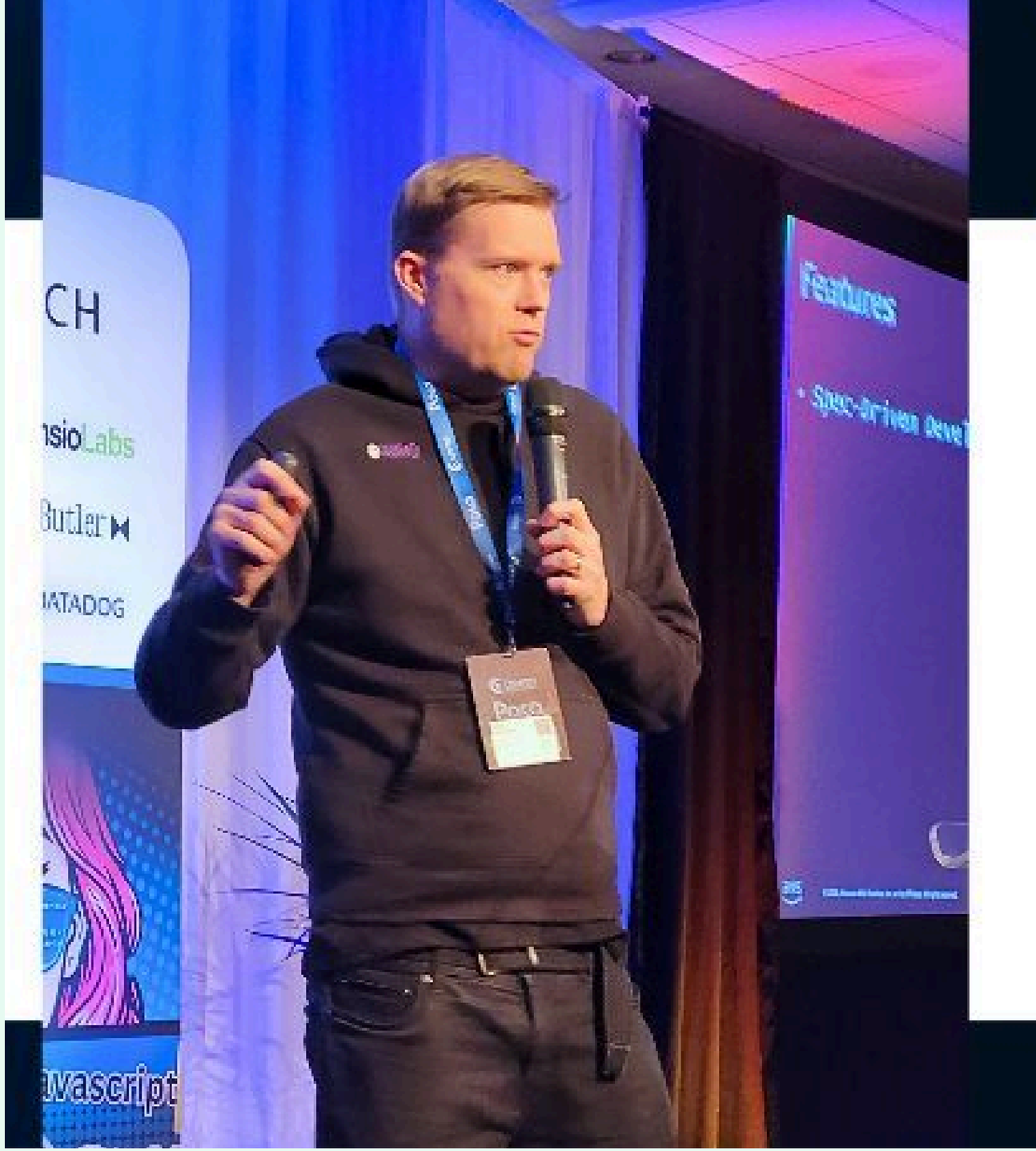
Embracing the Age of AI Tools

Transitioning from writing every line of code to guiding **intelligent agents** has transformed my role. Now, I focus on **orchestration** and creativity more than typing alone.

“Your expertise and taste is **MORE valuable**, then ever right now”

Meet Erik Hanchett

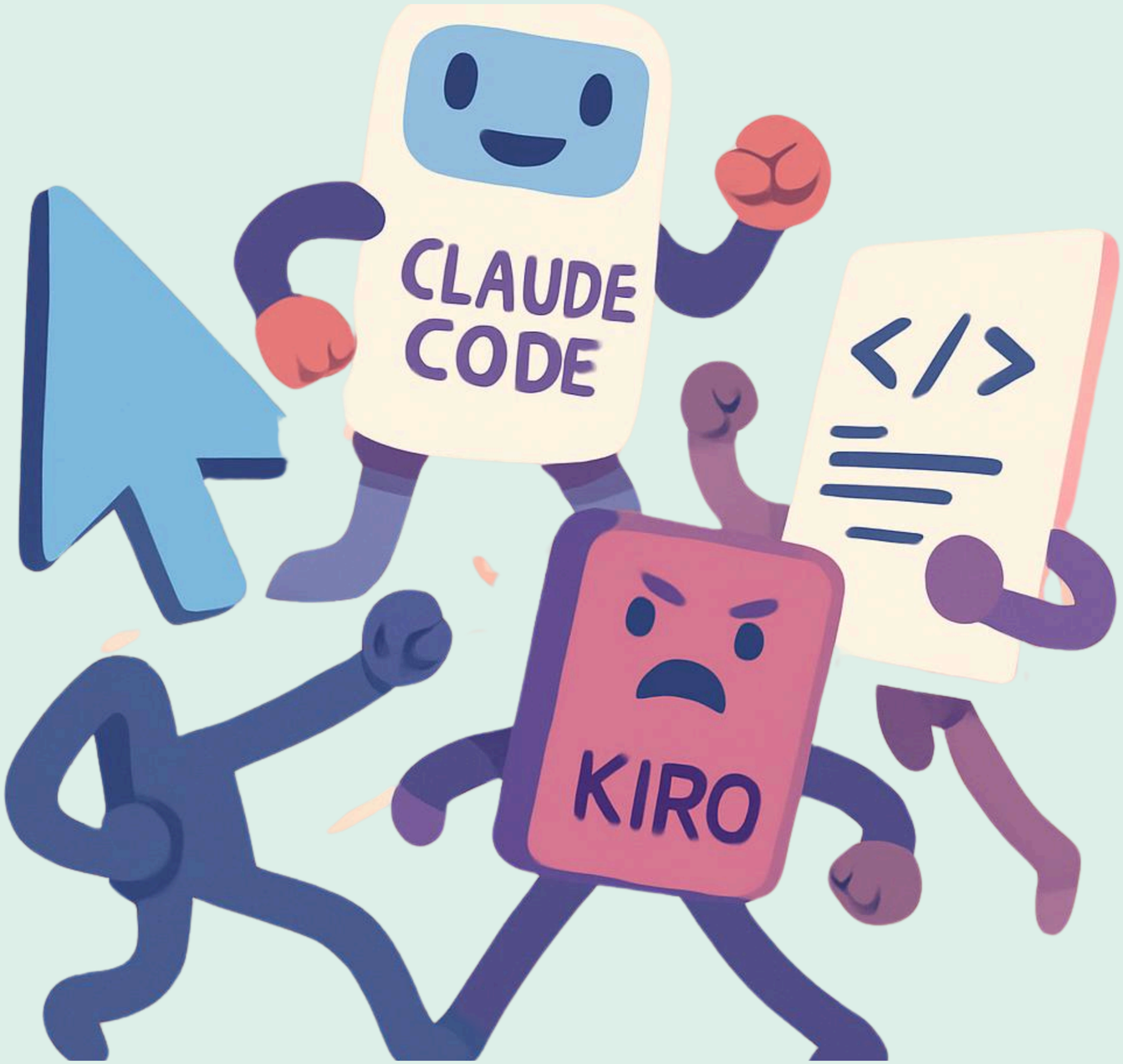
Erik Hanchett is a **Senior Developer Advocate** at AWS with over 15 years of experience in software development. He is the creator of Kiro content and the "Program with Erik" series, focusing on modern development practices.



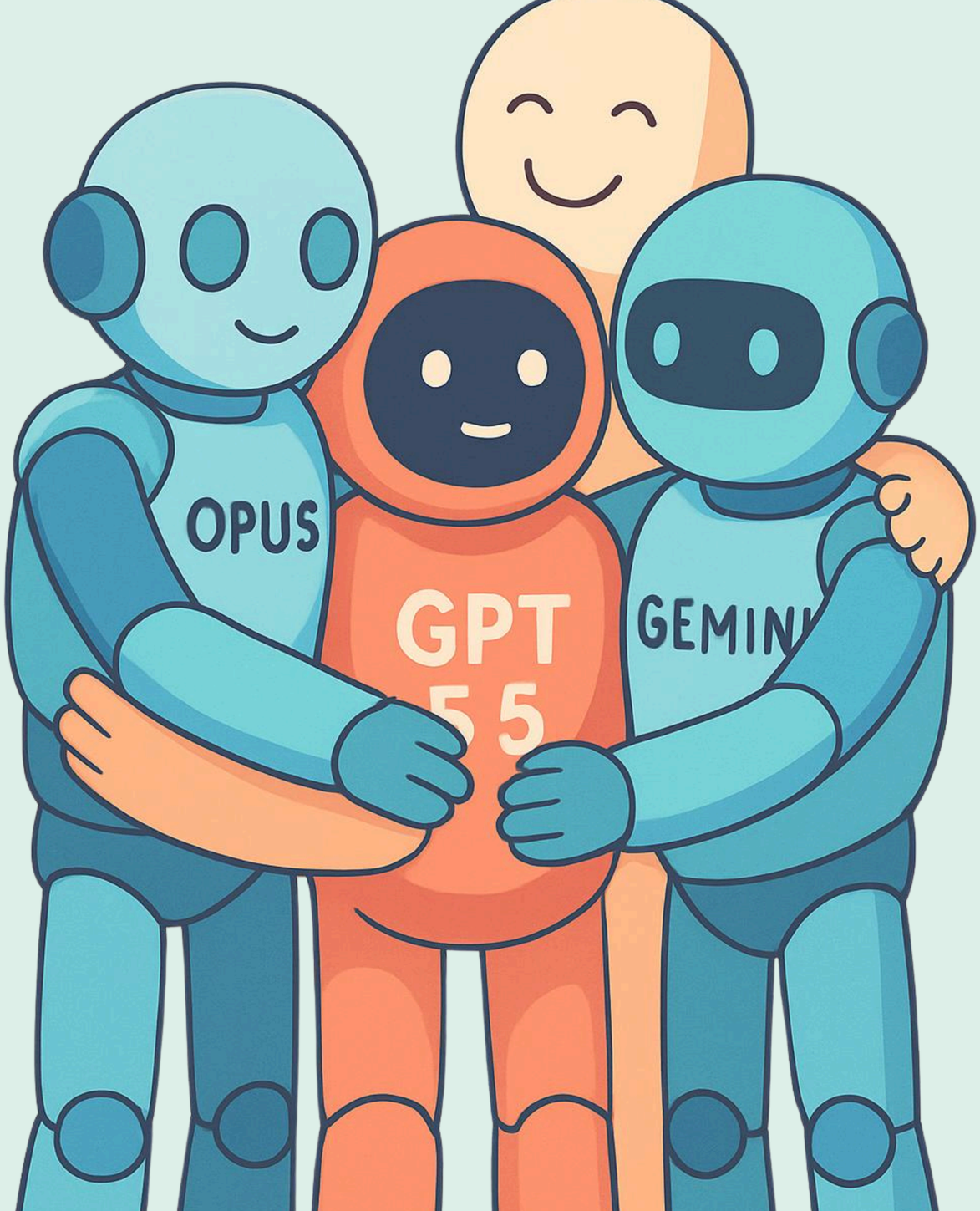
Where to Begin?

Pick a Harness and Model

Kiro
Claude Code
Codex
Cursor



Pick a frontier model



CLI > UI

- Use your IDE as your viewer , the CLI as your agent

Typing > Whisper

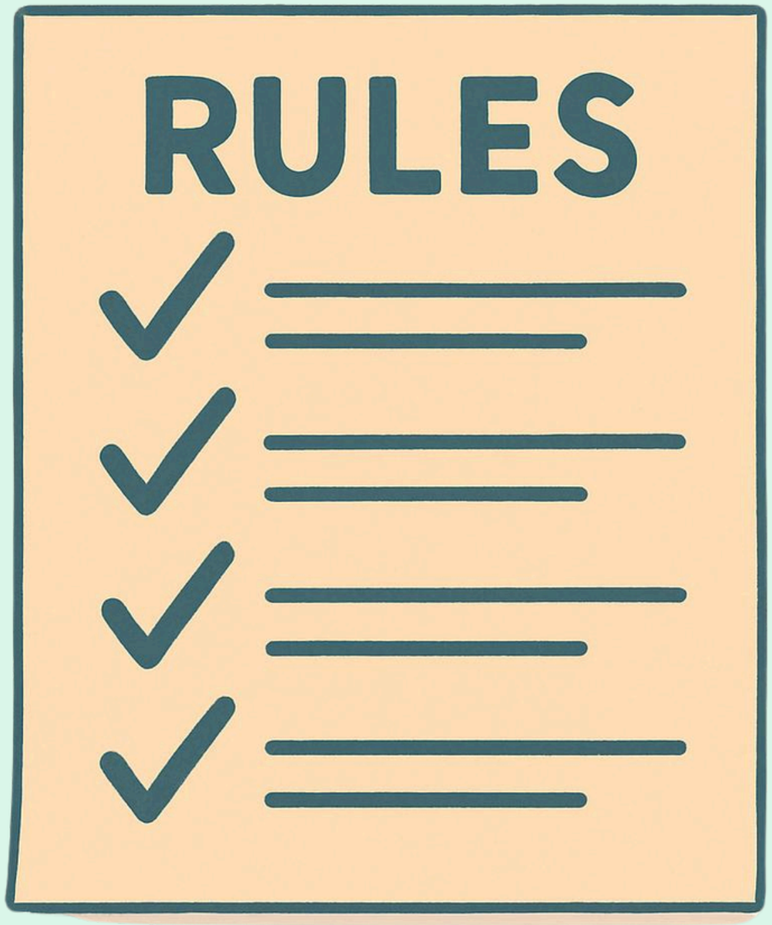
- **Less vocal strain**

Create rules for the AI

LLMs operate statelessly, requiring you to communicate your standards anew each session. Use AGENTS.md, CLAUDE.md, or .cursorrules to maintain clarity and consistency.

Create rules (AGENTS.md)

- **Project’s purpose**
- **Architecture**
- **Coding standards**
- **Keep it brief**



AGENTS.md > # Vue Project — AI Rules > ## Vue Standards

```
# Vue Project — AI Rules

## Project

This is a Vue 3 + Nuxt application using TypeScript, Pinia, and Vue Router.
Use Composition API with <script setup lang="ts"> everywhere.

## Vue Standards

- ALWAYS use Composition API with `<script setup lang="ts">` — never Options API
- Prefer `ref` over `reactive` (reactive breaks on reassignment and destructuring)
- Use `shallowRef` for large objects and arrays — do not use deep reactivity when it's not needed
- Never destructure reactive props — use `toRef` or `toValue` instead
- Use `async/await` in composables, never `.then()` chains
- Use `computed` for derived state, never recompute in templates
- Keep components focused: split when a component has more than one clear responsibility
- Props down, events up — make data flow explicit with typed `defineProps` and `defineEmits`

## Nuxt Specifics

- Use Nuxt layers to keep features modular — related components, composables, and stores belong together in
a layer
- Keep component files small and scoped so parallel agent work doesn't cause conflicts
- Use Nuxt auto-imports — do not manually import Vue or Nuxt composables
- Use `useFetch` / `useAsyncData` for data fetching, not raw `fetch`

## TypeScript

- TypeScript strict mode always
- Type all props, emits, and composable return values explicitly
- Run `vue-tsc` before declaring a task done — type errors catch AI hallucinations

## Testing & Quality

- Vitest for unit tests — write tests alongside implementation, not after
- ESLint with `vue/recommended` rules
```

Use Skills

- **npx skills add <owner/repo>**

vue-skills

Agent skills for Vue 3 development.

> 🚧 **Early Experiment / Community Project**
>
> This repository is an early experiment in creating specialized skills for AI agents to enhance Vue 3 development. Skills are derived from real-world issues but may be incomplete due to hallucinations—please give feedback. If valuable, I plan to propose transferring this project to the Vue organization to benefit the wider community.

https://github.com/vuejs-ai/skills

Vue Best Practices Workflow

Use this skill as an instruction set. Follow the workflow in order unless the user explicitly asks for a different order

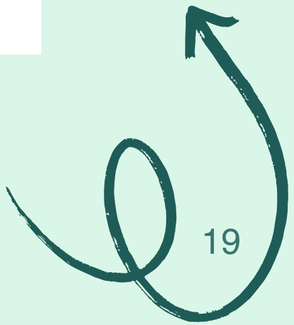
Core Principles

- **Keep state predictable:** one source of truth, derive everything else.
- **Make data flow explicit:** Props down, Events up for most cases.
- **Favor small, focused components:** easier to test, reuse, and maintain.
- **Avoid unnecessary re-renders:** use computed properties and watchers wisely.
- **Readability counts:** write clear, self-documenting code.

1) Confirm architecture before coding (required)

- Default stack: Vue 3 + Composition API + `<script setup lang="ts">`.
- If the project explicitly uses Options API, load `vue-options-api-best-practices` skill if available.
- If the project explicitly uses JSX, load `vue-jsx-best-practices` skill if available.

https://github.com/vuejs-ai/skills



Anthony Fu's Skills

A curated collection of Agent Skills reflecting Anthony Fu's preferences, experience, and best practices, along with usage documentation for the tools.

> **Important**
>
> This is a proof-of-concept project for generating agent skills from source documentation and keeping them in sync. I haven't fully tested how well the skills perform in practice, so feedback and contributions are greatly welcome.

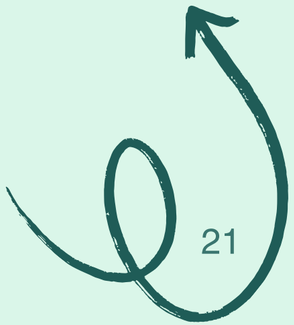
https://github.com/antfu/skills

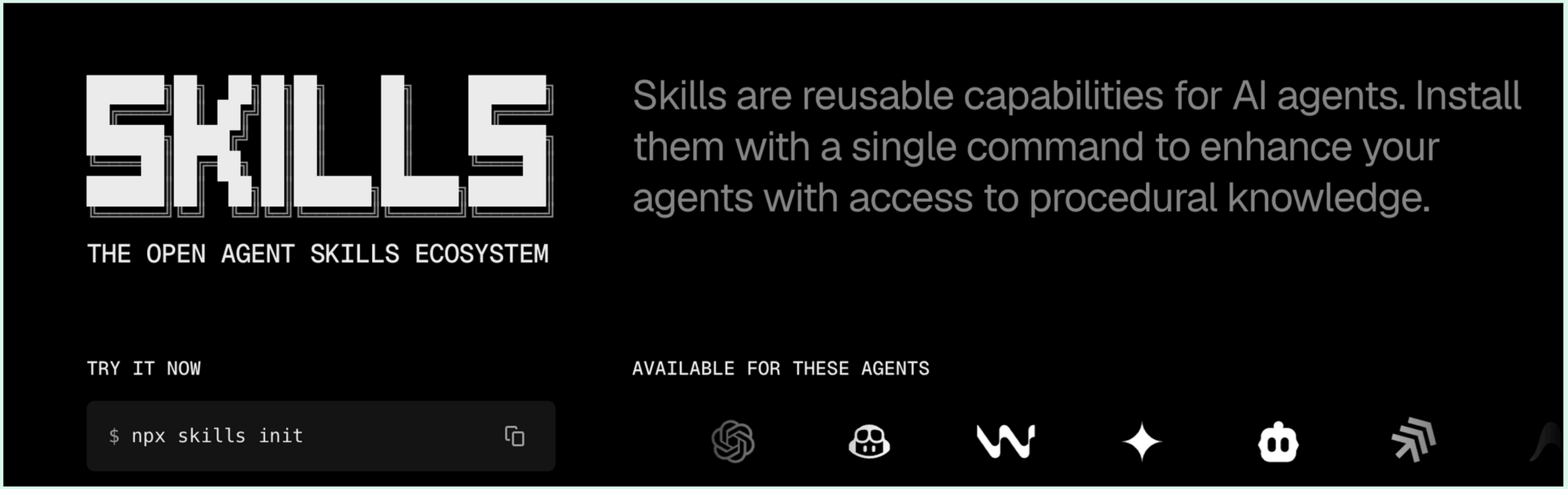
skilld

npm v2.0.0 downloads 5.9k/month license MIT

> Generate AI agent skills from your NPM dependencies.

https://github.com/skilld-dev/skilld

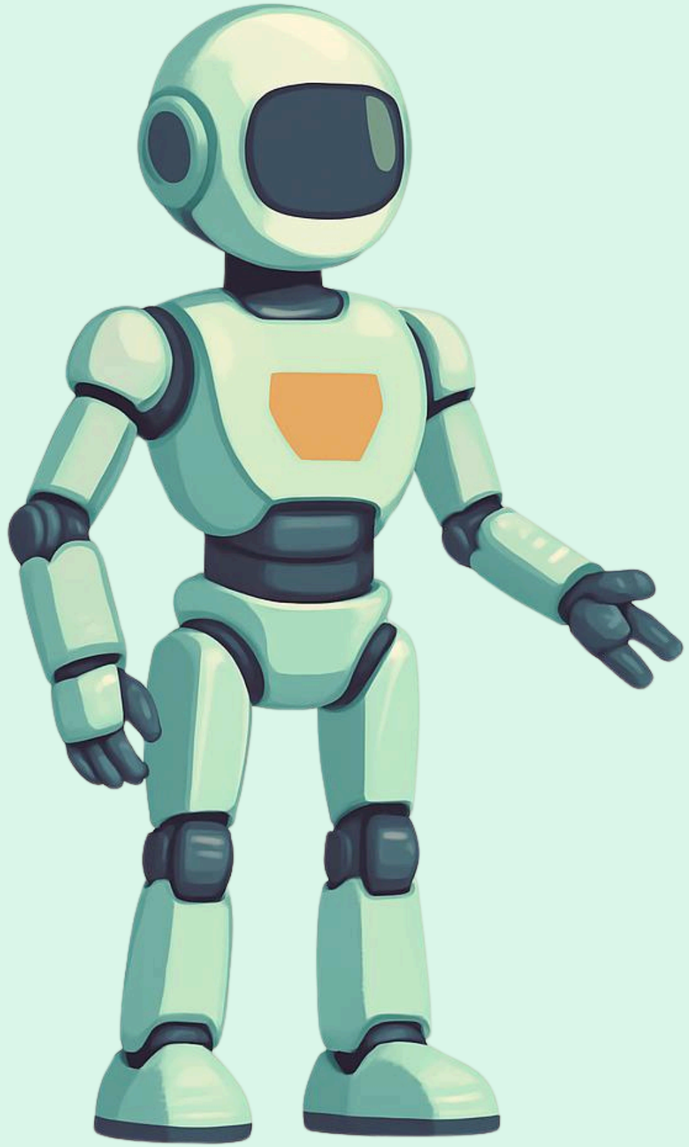




https://www.skills.sh/

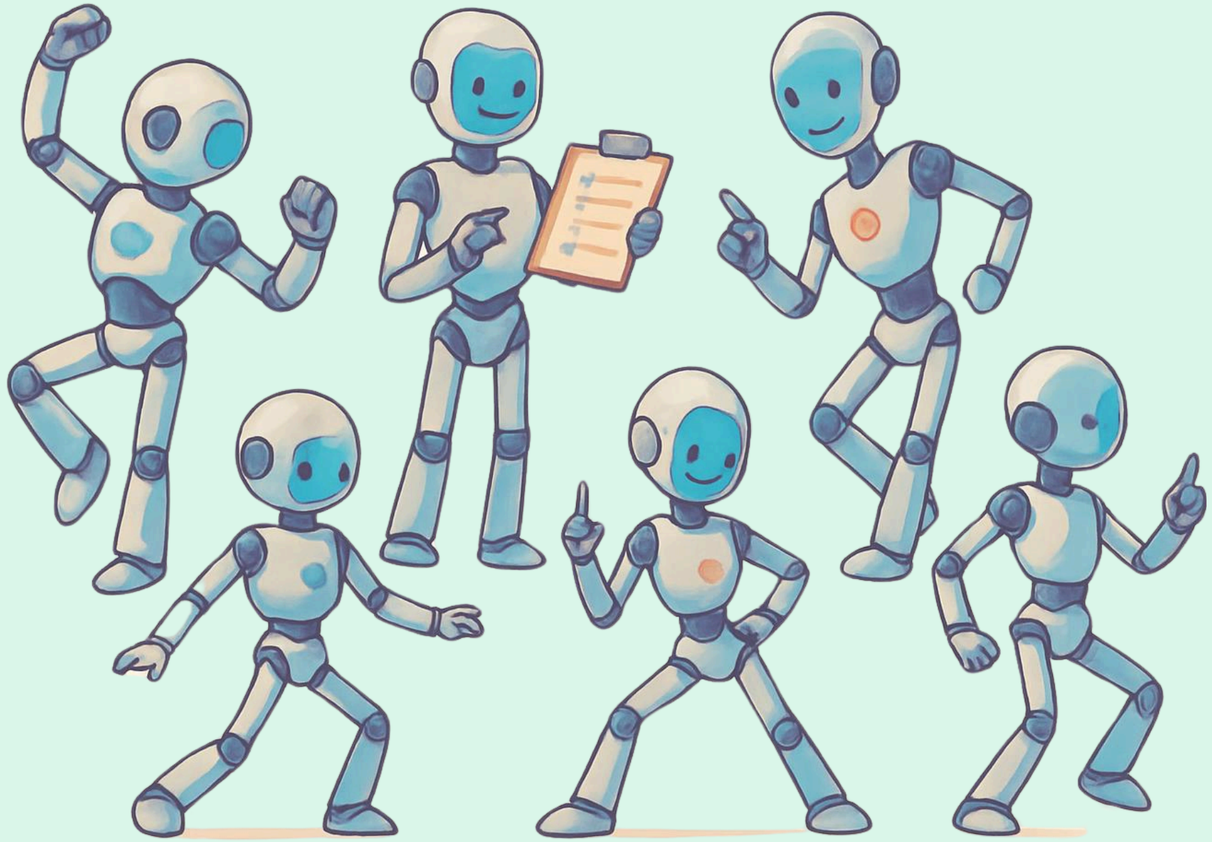
Create agents

- **/agents or /plugin**



So what can they do?

- Context
- Skill(s)
- Hooks
- Subagents
- Allowed Tools
- Autonomous



Setup an Agent Workflow

- (Research / Requirements / Design / Implementation)
- Implement
- Test

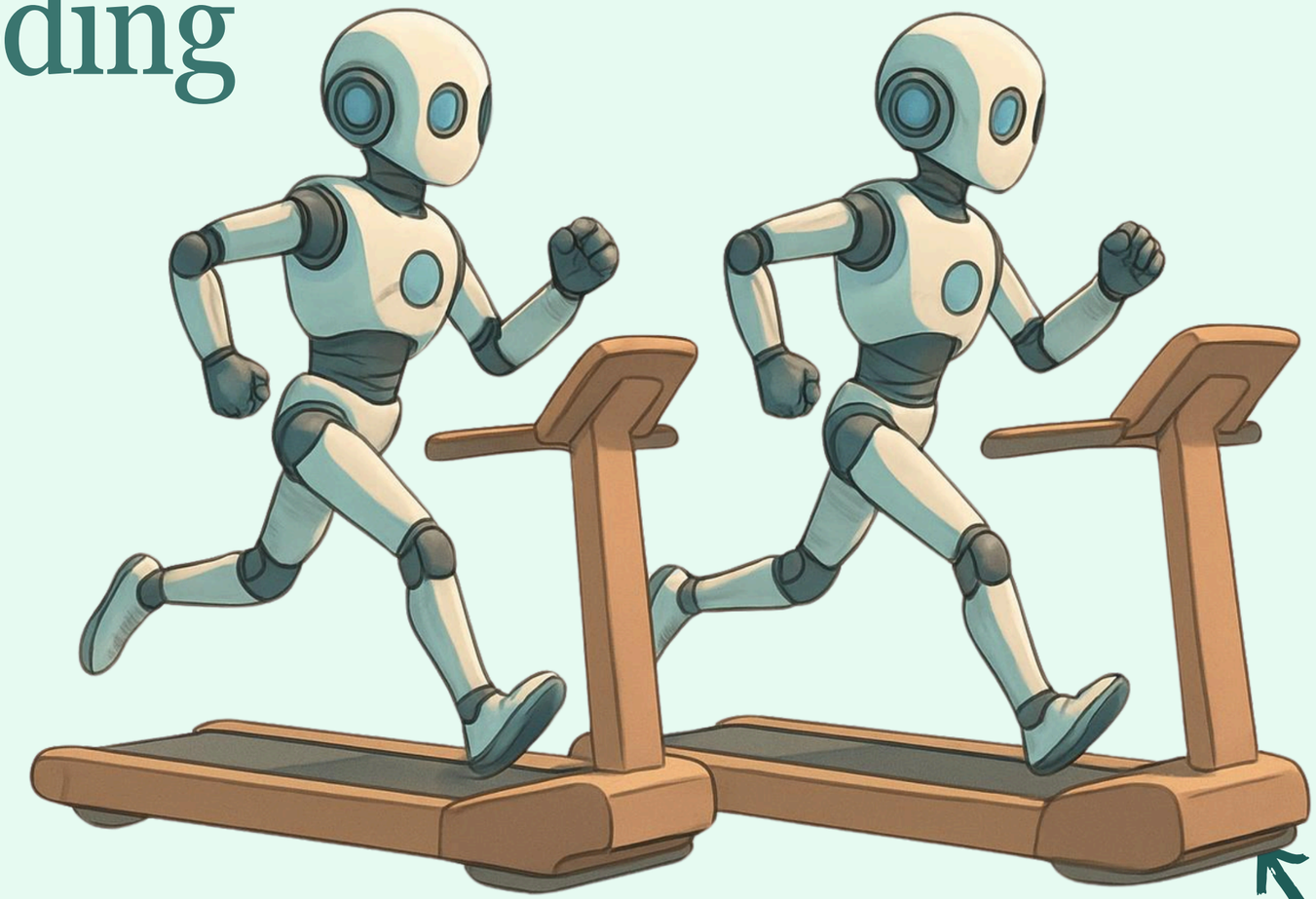
Setup an Agent Workflow

https://github.com/VoltAgent/awesome-claude-code-subagents

Setup an Agent Workflow

- **Build your own, use skills to help build agents and skills**

Working with Coding agents/tools



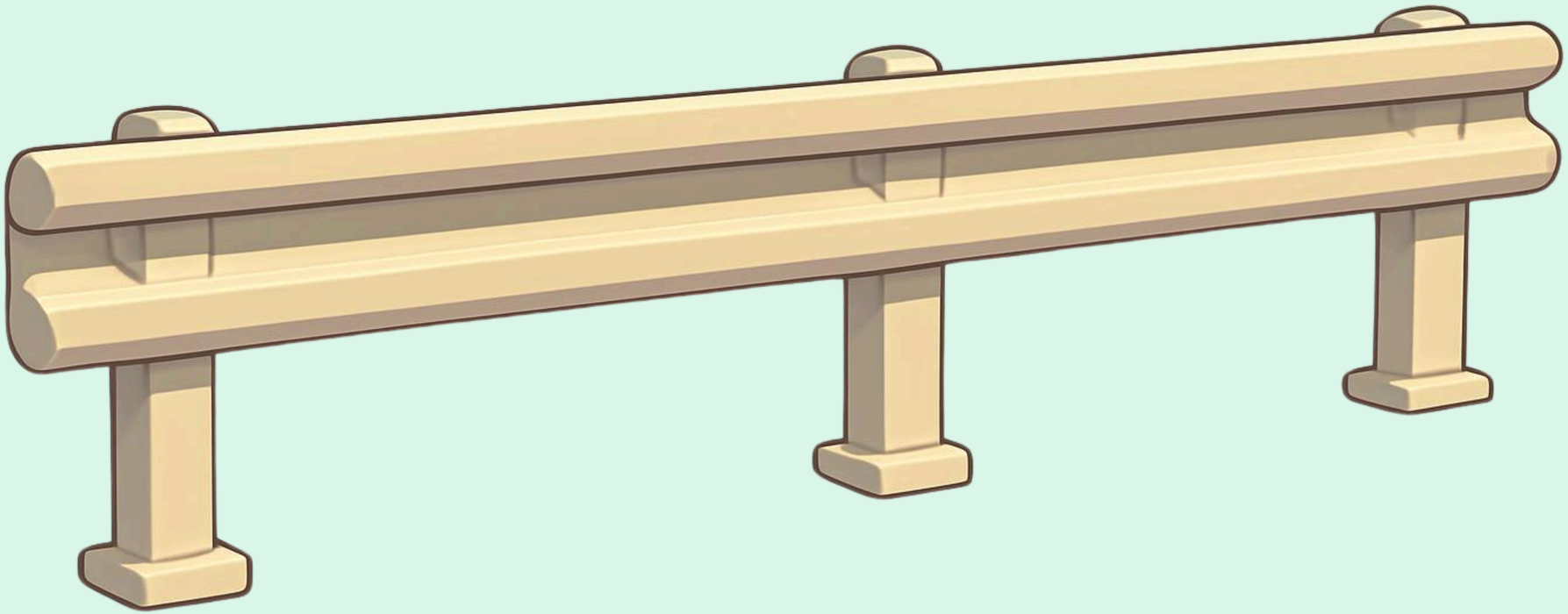
Two Coding Assistants

You can effectively run **two simultaneously** to increase productivity. This allows for parallel coding tasks.

File Management

Keeping tasks **small and scoped** is essential.
This approach helps prevent conflicts.

Setup Guardrails

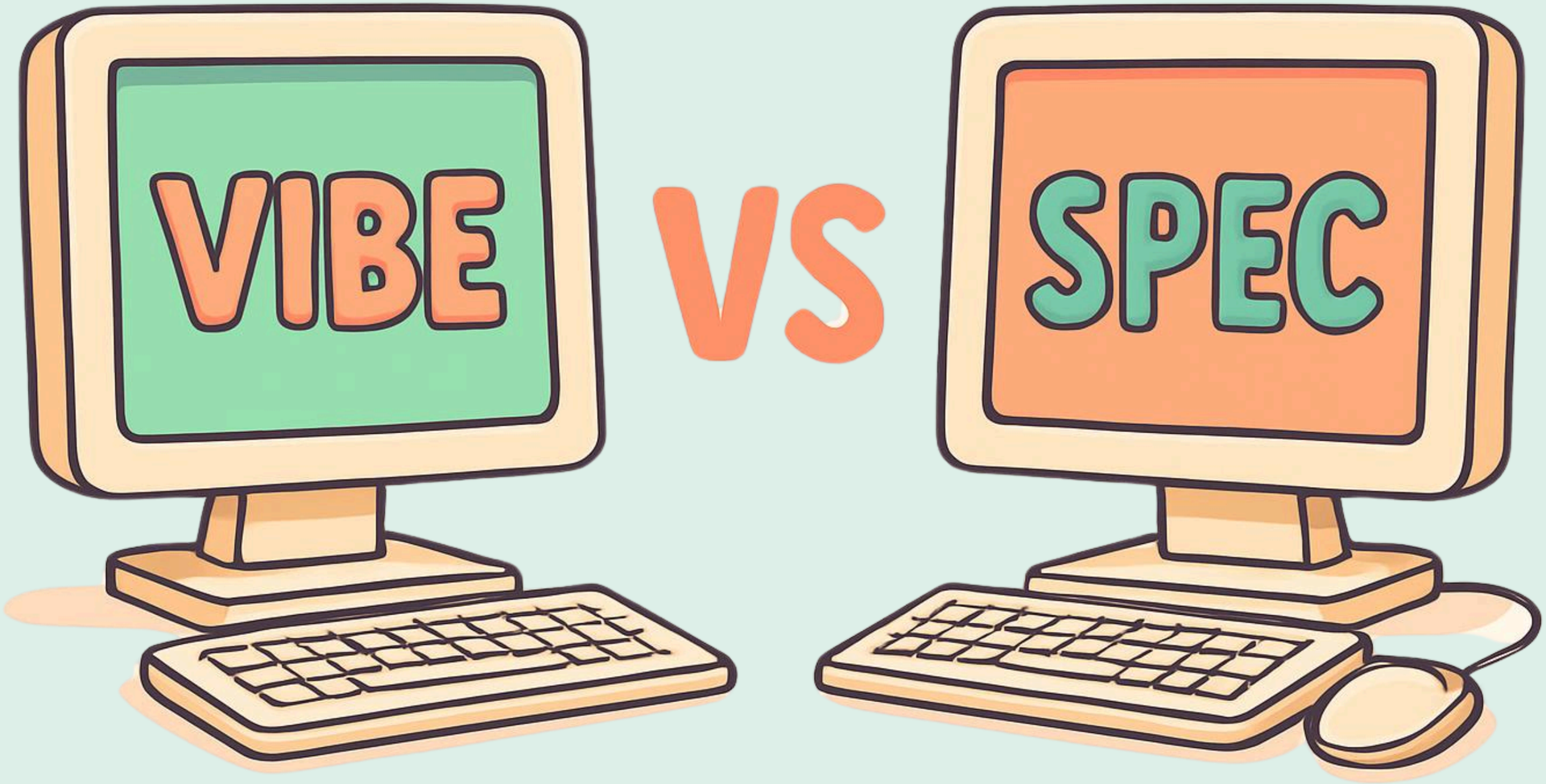


Guardrails

- Tests /Vitest
- Linting
- CI Checks
- Code Coverage
- Error Tracking
- Pre-commit hooks

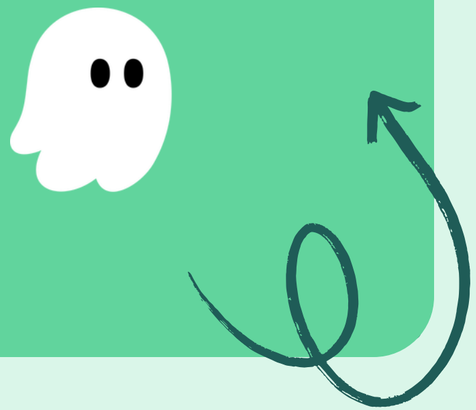
Code Reviews Are Bottlenecks

AI creates faster than we can review


VIBE
VS
SPEC

Spec-Driven Development

Give AI a blueprint instead of what you want!



The Three Phases

Structuring Your Development Workflow

Requirements

The first phase is about clearly defining **what to build**. Gather input from stakeholders to ensure all needs are understood and documented in a structured manner.

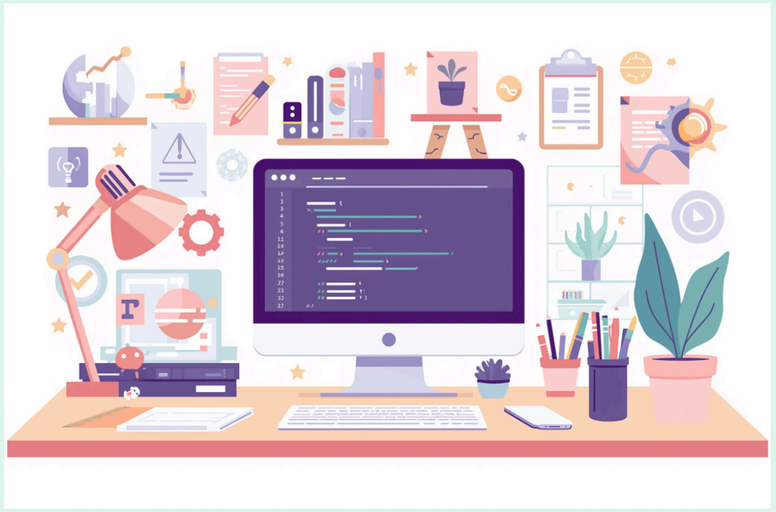
Design

In the design phase, outline **how to build** the solution. Create wireframes and architectural diagrams to visualize the project, ensuring alignment with the requirements established earlier.

Implementation

Finally, during implementation, convert designs into functional code. This phase focuses on translating requirements and designs into a working product, ensuring quality through best practices.

Essential Tools for Spec-Driven Development



KIRO

Streamlines spec-driven workflows for Vue developers

SPEC-KIT

Open-source templates to enhance your specs

CLAUDE CODE / OTHER

Has a plan mode

Speeding Up Development Processes

The **Quick Plan** generates a full spec from just a few questions, while parallel execution allows tasks to run simultaneously, making implementation nearly four times faster and catching logical conflicts early.

Vibe Coding

- Easy to start with
- Fast prototyping
- Fixing bugs
- Small scoped tasks

Spec Driven

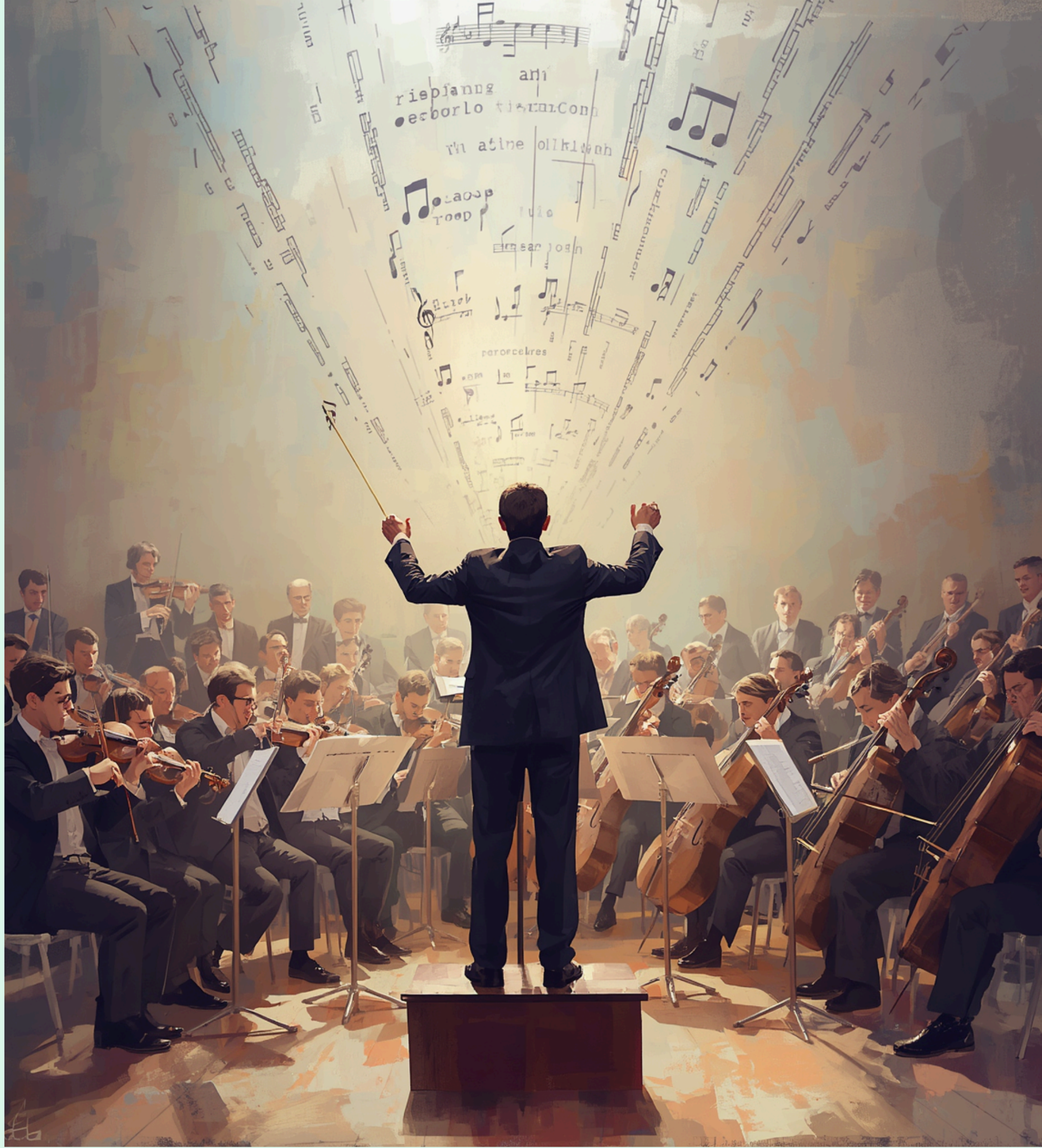
- Good for feature development
- Scope is large or unclear
- Need to parallelize
- Scales well for complex features

Live Demo of Kiro Flow

Key Takeaways for Developers

- Pick a frontier Model
- Choose a Harness (I prefer Kiro/Kiro CLI)
- Prefer CLI over UI
- Run no more then two executions at a time
- Create an AGENTS.md
- Use Skills
- Create Agents
- Setup Guardrails (Automated code reviews)
- Try out Spec-Driven Development

Embracing Your New Role



Essential Resources

Here are key resources to enhance your Vue development skills: explore **kiro.dev** for structured workflows, visit **github.com/github/spec-kit** for templates, and join the community at **discord.gg/kirodotdev** for support.

Thank You!

Connect with me on LinkedIn
Erik Hanchett