Process AI

OpenSpec vs Spec Kit: Choosing the Right AI-Driven Development Workflow for Your Team

by Vinicius Negrisolo on October 21, 2025

After exploring Spec Kit in my previous posts about spec-driven development, I decided to put OpenSpec to the test. Using the same task and starting from the same git commit, I wanted to see how these two AI-powered workflows compare in real-world usage.

Installation: A Developer-Friendly Start

Right off the bat, OpenSpec gave me a smoother onboarding experience. As someone who doesn't work with Python regularly, Spec Kit required me to install a newer package manager just to get started. OpenSpec, written in TypeScript, was as simple `npm install`. Then we initialize the project by running:

```shell
openspec init
```

During the init process, OpenSpec asks which AI tooling you're using. I chose Claude Code, but they offer several other options as well.

The installation added just **3 AI commands** into Claude Code, compared to the 8 generated by Spec Kit:

- `/openspec:proposal` - Scaffold a new OpenSpec change and validate strictly
- `/openspec:apply` - Implement an approved OpenSpec change and keep tasks in sync
- `/openspec:archive` - Archive a deployed OpenSpec change and update specs

The initialization created only two files: - `openspec/project.md` - `openspec/AGENTS.md`

There's no `/speckit.constitution` equivalent step. Instead, OpenSpec suggested:

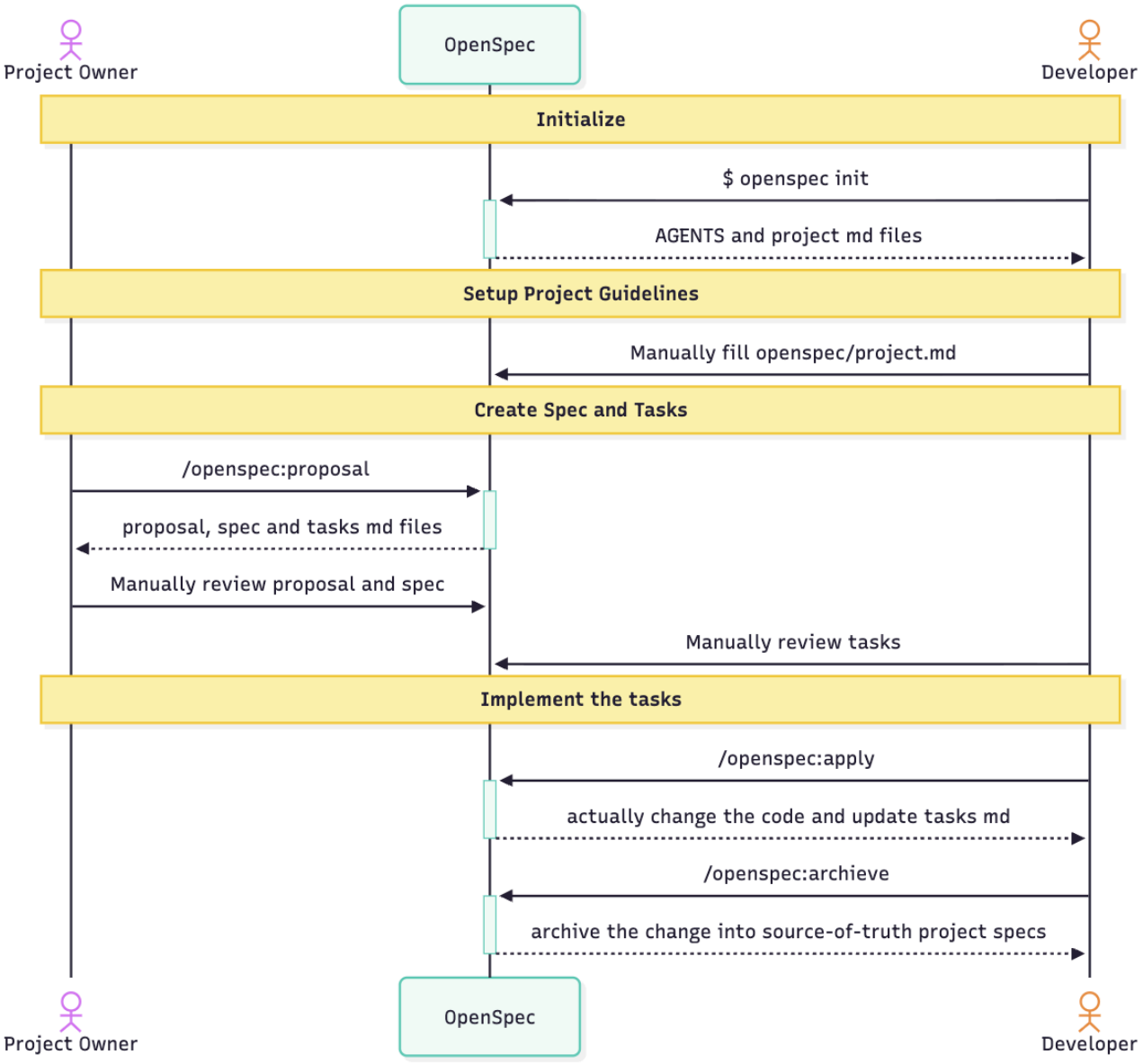
> "Please read openspec/project.md and help me fill it out with details about my project, tech stack, and conventions"

At this point I simply asked to the AI to introspect my project and fill out `openspec/project.md` . This produced a concise 250-line markdown file with comprehensive project details - the same size as Spec Kit's constitution output. My prompt was as simple as:

```plaintext
introspect my current project and fill openspec/project.md
```

Understanding the OpenSpec Flow

Before diving into my first spec, here's the OpenSpec sequence diagram to visualize the workflow:



You can also compare to a similar Spec Kit sequence diagram that I've created in the last post.

Creating My First Spec

I used the exact same prompt from my Spec Kit experiment:

```
PLAINTEXT
/openspec:proposal Remove the Team Tab from top of website. Leave it
```

OpenSpec did log some validation errors during generation, which was slightly concerning but didn't block progress. The execution generated 3 files totaling about **250 lines**:

- `openspec/changes/remove-team-nav/proposal.md`
- `openspec/changes/remove-team-nav/specs/navigation/spec.md`
- `openspec/changes/remove-team-nav/tasks.md`

Compare this to Spec Kit's **800 lines** at the same stage. The reduced verbosity makes OpenSpec's output significantly easier to review and digest.

One notable difference: OpenSpec doesn't automatically create a git branch like Spec Kit does. While this requires a manual step, it also gives you control over branching strategy.

Implementation: Fast and Focused

With OpenSpec, there's no separate task breakdown phase - you go straight from proposal to implementation:

PLAINTEXT

```
/openspec:apply
```

This execution was noticeably faster than Spec Kit's implementation phase, likely due to less noise in the spec files and fewer tasks to process. The results were impressive:

MARKDOWN

```
**Changes Made:**
1. Removed the Team navigation item from `app/views/layouts/_header.h
2. Verified footer navigation still contains Team link
3. Confirmed Rails application loads successfully
4. Checked for CSS dependencies (nth-child selectors)

**Test Results:**
- RSpec: 327/328 passing (1 pre-existing failure unrelated)
- Cucumber: 54/59 passing (5 pre-existing failures unrelated)
- The AI even identified our flaky tests and confirmed they were unre
```

The implementation checked for CSS nth-child selectors that could break - a detail that's easy to overlook but could cause subtle bugs.

Archiving: Building Your Knowledge Base

If we check the OpenSpec dashboard now we have:

```shell
$ openspec view

OpenSpec Dashboard

═══════════════════════════════════════════════════════════════

Summary:
  • Specifications: 0 specs, 0 requirements
  • Active Changes: 0 in progress
  • Completed Changes: 1

Completed Changes
─────────────────────────────────────────────────────────────

  ✓ remove-team-nav

═══════════════════════════════════════════════════════════════
```

At this point, the change is implemented but not yet considered a "spec" in the dashboard. To consolidate the knowledge, run:

```plaintext
/openspec:archive
```

This step moves the completed change from `changes/` to `changes/archive/` and consolidates the acquired knowledge into permanent specifications. In this case, it created `openspec/specs/navigation/spec.md`.

After archiving, the dashboard reflects the consolidated knowledge:

```shell
$ openspec view

OpenSpec Dashboard

═══════════════════════════════════════════════════════════════
```

```
Summary:
  • Specifications: 1 specs, 3 requirements
  • Active Changes: 0 in progress
  • Completed Changes: 0

Specifications
──────────────────────────────────────────────────────────────────

  ▪ navigation                          3 requirements

══════════════════════════════════════════════════════════════════
```

The `AGENTS.md` file created in the begining by `openspec init` command does include a reminder to check these consolidated specs:

MARKDOWN

```
## Before Any Task

**Context Checklist:**
- [ ] Read relevant specs in `specs/[capability]/spec.md`
```

This means future work will benefit from the knowledge captured in this workflow.

AI Tooling Trade-offs

AI tools are non-deterministic so it's very hard to compare them properly. On the top of that the input that I gave to the AI tool was a bit poor in refinement and I did chose to continue with the flow for speed intead of reviewing each interaction properly. Also the change that I asked was a very simple change in the codebase, I was not trying to test the AI capabilities but I wanted to run the flows from begining to end in order to learn more.

Which Tool is Right for Your Team?

The choice between OpenSpec and Spec Kit depends heavily on your team structure and needs:

Choose OpenSpec if:

- You have senior developers who don't need hand-holding
- You work in small teams where roles might overlap
- You value conciseness and speed over comprehensive documentation

Choose Spec Kit if:

- You need clear separation between Product Owner and Developer roles
- You want extensive documentation and verification steps
- Your team includes junior developers who benefit from detailed guidance
- You prefer a more structured, prescriptive workflow

The reality is that modern AI-driven development workflows are still evolving. Both OpenSpec and Spec Kit represent slightly different philosophies about how to integrate AI into the software development process. Your choice should align with your team's experience level, size, and preferred working style.

Ready to Accelerate Your Development with AI?

Whether you choose OpenSpec, Spec Kit, or another AI-driven workflow, the key is having a team that knows how to leverage these tools effectively. At **Hashrocket**, we specialize in building high-quality web and mobile applications using modern technologies and AI-powered productivity tools.

Our expertise includes:

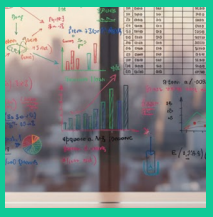
- **Elixir & Phoenix** for scalable, real-time applications
- **Ruby on Rails** for rapid, reliable web development
- **React & React Native** for responsive web and mobile experiences
- **AI-Integrated Workflows** to ship features faster without sacrificing quality

Need help building your next project or integrating AI tools into your development process? Let's talk about how we can help your team to ship better software.

Was this post helpful? Share it with others.

If you enjoyed this post, check out these related articles next:

- How To Rev Up Your Rails Development with MCP
- Building a MCP Server in Elixir
- Some Thoughts About Claude Code



Up next

Spec-Driven Estimation: How Devs Can Estimate Features Quickly and More Accurately with AI

More posts about Process Development AI

ERICSSON

We're proud to have launched hundreds of products for clients such as LensRentals.com, Engine Yard, Verisign, ParkWhiz, and Regions Bank, to name a few.

Let's talk about your project

Subscribe Today!

Stay ahead of the curve. Receive valuable blog posts, resources and event notices right to your inbox.

Email address

Sign Up

Jacksonville Beach

Chicago

320 1st Street N #711
Jacksonville Beach, FL 32250

661 W Lake St. Suite 3NE
Chicago, IL 60661

1-904-339-7047

Facebook Twitter Github