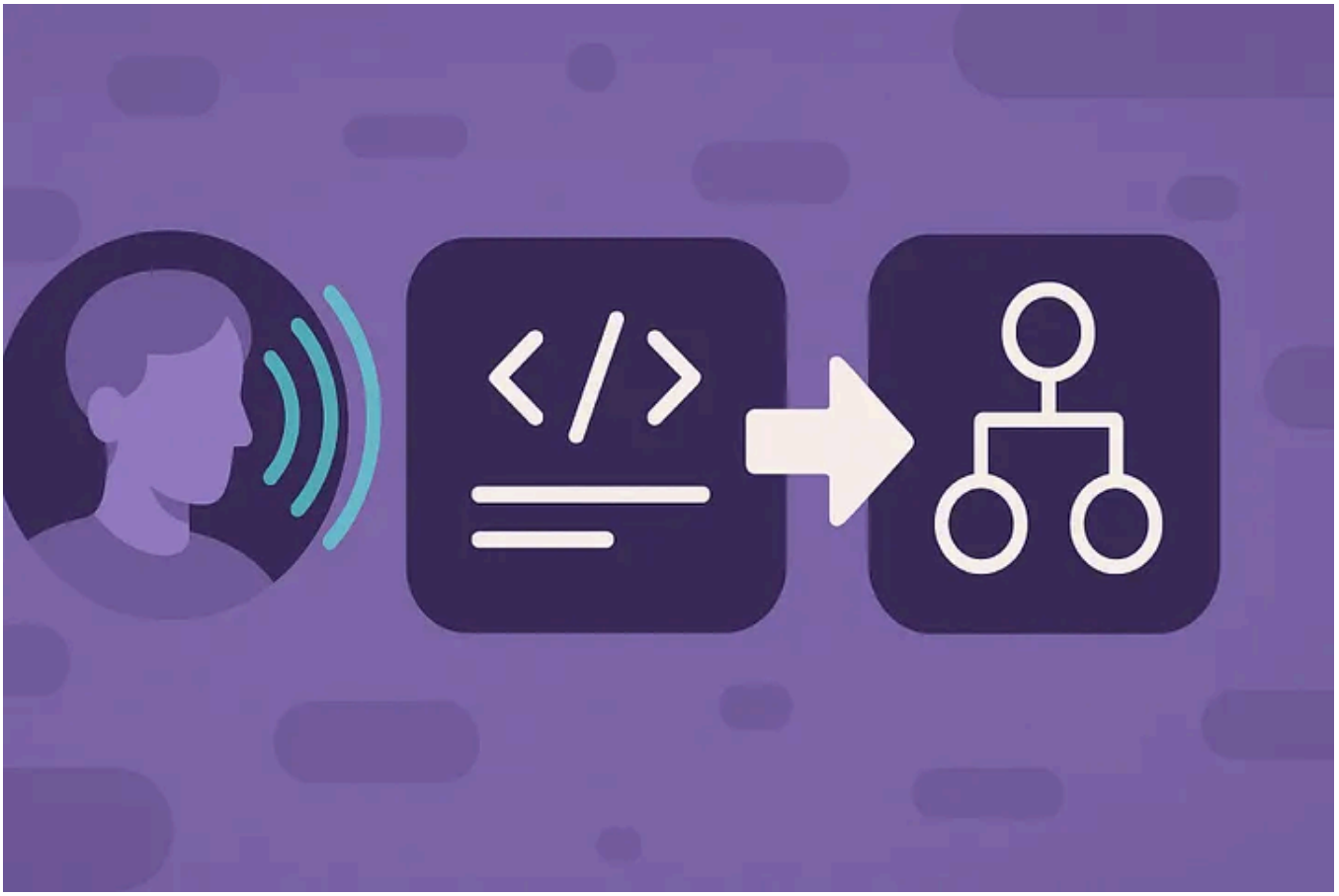
Spec Driven Development (SDD): The Evolution Beyond Vibe Coding

14 min read · Sep 16, 2025

Daniel Sogl Follow



Since “vibe coding” emerged in early 2025 — coined by Andrej Karpathy on February 2nd — developers have been able to create full applications through AI-assisted coding tools. The numbers are staggering: 25% of Y Combinator’s Winter 2025 cohort have codebases that are 95% AI-generated. Initially, there was a boom in creating applications by typing basic prompts into chat windows. While the results looked

impressive — though often generic and built on the same UI libraries — developers grew frustrated over time.

LLMs excel at generating new content, but they often fall short on code quality, architecture choices, API patterns, and security standards. This isn't because LLMs lack capability compared to professional developers, but because developers struggle to define exactly what they want in their prompts, how the LLM should achieve the goal, and how results should be validated. It turns out that while developers are efficient at writing code, they often fail to formulate exactly what they want in plain text.

When I started learning software development in 2012, I encountered Test Driven Development (TDD), where developers define tests with expected results before writing actual code. I never fully adopted this technique in my development work, though I understood its merits. I found it impractical for daily work because for me, writing software is a creative process that evolves over time. While I consider architecture and patterns before starting, I typically begin coding and refactor later after implementing the requested feature. Writing tests beforehand would certainly help create better structured code from the start, but it would also slow down the creative process.

However, I once worked for a company developing a complex product where I learned the importance of defining user stories precisely, including personas and acceptance criteria. I often found that stories didn't fully reflect what the product owner wanted me to implement, causing multiple iterations of coding, QA feedback, and adjustments.

With these lessons from the past and current frustrations with AI-assisted coding tools, there's now a promising pattern that merges the concepts of TDD and well-defined user stories while still allowing creativity in coding — with AI tools handling the heavy lifting.

Introduction to Spec Driven Development

When we examine what "Vibe Coding" truly means, we realize it lets us forget that code even exists. The main idea is simple: instead of defining goals as code, we express them in plain, natural language. Thanks to LLMs, even people without deep technical knowledge can bring their ideas to life. However, this approach typically works only for prototypes. Production-ready code shouldn't just look cool — it needs to be bug-free, architecturally sound, secure, and functional. To achieve this,

developers must define custom instructions with various rules, add MCP servers, and write precise prompts that provide clear guardrails for the LLM. Unfortunately, developers often underestimate the effort required for this setup and prompt engineering. This is where Spec Driven Development (SDD) comes in. Instead of coding first and documenting later, SDD begins with defining a specification. This specification serves as a contract for how your code should behave and becomes the source of truth that your tools and AI agents use to generate, test, and validate code. The result is less guesswork, fewer surprises, and higher-quality code.

Spec Driven Development Phases

SDD divides development into four distinct phases, each with specific goals and outcomes. Throughout these phases, we rely on our chosen AI coding tool for assistance. Our responsibility is to articulate our goals in natural language. The code that's eventually generated is merely the product of these clear specifications.

1. Specify

In this phase, you provide a high-level description of what you're building and why, then the coding agent generates a detailed specification. This isn't about technical stacks or application design — it's about user journeys, experiences, and success criteria. You'll define who will use your product, what problem it solves for them, how they'll interact with it, and which outcomes matter most. Think of it as mapping the desired user experience while letting the coding agent elaborate on the details. This specification becomes a living document that evolves as you gain deeper insights into your users and their needs.

2. Plan

This phase is where you get technical. You'll provide the coding agent with your desired tech stack, architecture, and constraints, and it will generate a comprehensive technical plan. If your company has standardized technologies, specify them here. Detail any integrations with legacy systems, compliance requirements, or performance targets you need to meet. You can also request multiple plan variations to compare different approaches. By making your internal documentation available to the coding agent, it can directly incorporate your architectural patterns and standards into the plan. Think of this as setting the rules before the game begins — the coding agent needs to understand the parameters before it can start working effectively.

3. Tasks

The coding agent transforms the spec and plan into actionable work by generating small, reviewable chunks that each address a specific component. Each task should be implementable and testable in isolation — this approach allows the coding agent to validate its work and maintain focus, similar to a test-driven development process. Rather than broad directives like “build authentication,” you receive precise tasks such as “create a user registration endpoint that validates email format.”

4. Implement

The coding agent executes tasks sequentially or in parallel as appropriate. The key difference here is that instead of reviewing massive code dumps, you — the developer — evaluate focused, specific changes that address particular problems. The coding agent has clear direction: the specification defines what to build, the plan outlines how to build it, and the task specifies exactly what to work on.

Crucially, your role extends beyond steering — you must verify. At each phase, reflect and refine. Does the specification truly capture your intended product? Does the plan address real-world constraints? Has the AI overlooked any edge cases or omissions? The process includes deliberate checkpoints where you evaluate what’s been generated, identify gaps, and correct course before proceeding. While the AI generates the artifacts, you ensure their accuracy and relevance.

Introduction to GitHub Spec Kit

Since following SDD requires defining extensive prose text — including specifications, tech stack details, test cases, and more — GitHub released an open-source CLI called Spec Kit to simplify creating the required files. Released on September 2, 2025, Spec Kit is currently at version 0.0.30+ and continues rapid development with frequent updates and community contributions.

Spec Kit Installation

Spec Kit can be installed in two different ways. The first method uses uvx, a Python package manager that works like npm with npx. I recommend installing uvx through Homebrew.

```
uvx --from git+https://github.com/github/spec-kit.git specify init <PROJECT_NAM
```

This command creates a new folder with the defined name. You'll then confirm which AI coding assistant you want to use. Currently, you can't switch between selected tools after initialization.

If you want to create the files in an existing project, you can pass the `--here` flag.

Alternative installation method: You can download platform-specific template packages without using the CLI. These are available on the release page in both POSIX shell (sh) and PowerShell (ps) variants.

If you want to create the files in an existing project, you can pass the `--here` flag.

Another option is to download and extract the zip file containing all necessary files without running the CLI. These zip files are available on the release page.

Project setup

To demonstrate Spec Kit, I'll create a new project using GitHub Copilot as my AI coding assistant. For my tech stack, I'll use Angular with the latest ESLint, Prettier, and TypeScript APIs. This will also show how to ensure Spec Kit follows the most current APIs of your preferred framework.

Spec Kit Files Overview

Based on your selected tool, Spec Kit creates various files in specific directories. Understanding this structure is crucial for effective implementation.

Prompt Files

The prompt folder contains reusable prompts that can be executed through GitHub Copilot. These prompts include instructions that GitHub Copilot should follow. We will use these prompts to create our initial specification, technical implementation plan, and finally, the tasks we can execute step by step. This approach simplifies following Spec Driven Development guidelines, as the GitHub team has precisely defined what the AI agent must do, ensuring it follows the established guardrails.

Memory Directory

The `memory/` folder contains the constitution, which defines your principles such as preferred architecture, API exposure patterns, TDD approach, libraries, technologies, and coding standards. You should fill out this file in detail before running any prompts. Your AI coding tool can help with this process. I recommend using context7 MCP or another documentation MCP server to retrieve the latest style

guides for your framework. For best results, use the most advanced LLMs available, such as Sonnet 4 or GPT-4.

```
Fill out the project constitution for a modern Angular web application. Ensure
```

Scripts Directory

The `scripts/` directory contains shell scripts executed by the coding agent to help it understand specifications, create branches, and generate other required files. You should not modify these files manually, as they're integral to the SDD workflow automation.

AI Tool Integration

Different AI tools use different instruction files:

- **GitHub Copilot:** `.github/copilot-instructions.md`
- **Cursor:** `.cursorrules` files
- **Claude:** `CLAUDE.md`
- **Gemini:** `GEMINI.md`

Each file contains tool-specific prompts and instructions that guide the AI through the SDD phases.

Creating Your First Specification

After defining the constitution, it's time to create your first specification. This initial step is crucial for defining your application's goal. Be as specific as possible. You can also use your preferred AI to enhance the description. Focus on the user experience rather than technical details.

To use the specify functionality, execute the command through your chosen AI tool. In GitHub Copilot's case, this involves using the `/specify` command:

```
/specify NG Pokedex is an application designed for Pokémon fans who want a simp

When users want to dive deeper, they can click on a Pokémon to open a detail vi
```

```
The experience is designed to be seamless across devices—desktop, tablet, and m
```

Running this prompt creates your first spec file and branch. For each specification, the agent creates a separate branch to work on — in my example, `001-ng-pokedex-is`. This process may take a few minutes, as the coding agent calls multiple scripts, creates files, and generates user stories, requirements, and key entities. The agent ensures it follows all instructions defined by the Spec Kit CLI.

Open the generated `spec.md` file and scroll through it. At the bottom of the file, the agent validates the following requirements:

```
### Content Quality
- [x] No implementation details (languages, frameworks, APIs)
- [x] Focused on user value and business needs
- [x] Written for non-technical stakeholders
- [x] All mandatory sections completed

### Requirement Completeness
- [x] No [NEEDS CLARIFICATION] markers remain
- [x] Requirements are testable and unambiguous
- [x] Success criteria are measurable
- [x] Scope is clearly bounded
- [x] Dependencies and assumptions identified
```

If any of these checkboxes remain unchecked, you should either rerun the specify command with more detailed information or complete the document yourself. It's crucial to validate that the defined stories and requirements align with your vision for the application or feature. Edge cases, for example, need to be defined manually since the agent doesn't specify them automatically.

```
### Edge Cases

- What happens when no Pokémon match a search query? The system MUST display a
```

```
  - How does the system handle when a user tries to access detailed information f
  - What occurs when the user has no favorited Pokémon yet? The system MUST displ
```

Creating a Technical Plan

After finalizing the non-technical specification, it's time to create the technical plan for implementation. From my experience, you can optimize this process by establishing the default application structure upfront instead of relying on the coding agent to do so. I suggest using context7 MCP again to ensure you follow the latest project setup guidelines. In my case, I'll also incorporate the Angular CLI MCP server to generate the Angular project. Be prepared for this prompt to take several minutes to complete.

```
/plan The application will be developed using Angular 20 together with Angular
```

At this stage, the agent generates several files, including a **plan.md** document. This document outlines how the agent should research information needed to create tasks, and includes research results, instructions for following your guidelines, and entity model definitions. Review these files carefully to ensure the agent will adhere to your architecture, framework standards, and specification requirements. Pay particular attention to the **quickstart.md** file, as it contains instructions for initializing the project files. Verify that the npm versions match the latest versions of your preferred framework and tools.

Defining Implementation Tasks

This is the final specification step where Spec Kit helps us create task definitions for our chosen coding agent. This step doesn't require adding any specifications to the prompt file execution. Simply run the `/tasks` command in your chat window and wait while the coding agent defines all the tasks needed to implement your well-defined specification. Be patient, as this process can take several minutes to complete.

After executing the tasks prompt, the agent creates a **tasks.md** file, including a step-by-step definition of all tasks that need to be executed to create your defined specification. Again, review the file manually to make sure everything is defined and no todos are open at the end of the file.

```
## Validation Checklist
**GATE: All requirements met for execution**

- [✅] All 3 contracts have corresponding tests (T006-T008)
- [✅] All 9 entities have model tasks (T015-T023)
- [✅] All tests come before implementation (T006-T014 before T024+)
```

```
- [✅] Parallel tasks truly independent (different files, no shared state)
- [✅] Each task specifies exact file path with Angular conventions
- [✅] No task modifies same file as another [P] task
- [✅] Constitutional compliance: Standalone components, signals, Material Desi
- [✅] Domain boundaries respected: Pokemon, Favorites, Search
- [✅] Angular 20 modern patterns enforced throughout

**Total Tasks**: 44 tasks organized in 5 phases with clear dependencies
**Estimated Duration**: 3-4 weeks for full implementation with testing
**Parallel Opportunities**: 28 tasks marked [P] for parallel execution
```

The tasks are organized into different phases based on the specification's complexity. In my example, there are five phases: Setup, TDD, Core Implementation, Integration, and Polish. Each task has a unique identifier (e.g., T001) that you can reference when instructing your agent which task to complete first. It's essential to follow the phases and tasks in their intended order. If you complete a task manually, be sure to mark it as completed in the tasks file.

Now it's time for your coding agent to do the hard work for you!

Current Limitations and Considerations

As of September 2025, Spec Kit is still experimental (version 0.0.30+) and has some known limitations worth considering:

- **Tool Switching**: Once initialized with a specific AI tool, you cannot easily switch to another
- **Greenfield Focus**: Better suited for new projects than existing codebases
- **Python Dependency**: Requires Python 3.11+ for uvx installation
- **Rapid development**: Features and structure change frequently; documentation may lag behind latest capabilities
- **Community-driven expansion**: Many new AI tool integrations come from community pull requests, creating inconsistent quality

The Road Ahead

GitHub has positioned Spec Kit as an experiment in structured AI-assisted development. Early adoption statistics show promise — major tech companies are already incorporating SDD principles into their development workflows, and the

tool has gained traction among developers frustrated with unstructured AI coding approaches.

The integration ecosystem continues expanding. Recent updates to GitHub Copilot (including the new Coding Agent in public preview) and competitive responses from Cursor and other AI coding tools suggest that specification-driven approaches will become standard practice rather than experimental techniques.

Conclusion

Spec Driven Development promises a structured approach to AI-assisted coding that addresses the quality and maintainability issues inherent in "vibe coding." GitHub Spec Kit simplifies this process through predefined prompts and templates, providing developers with guardrails for AI code generation.

In theory, SDD appears to be the logical evolution of today's AI-assisted development, but in practice, it currently faces the fundamental challenge of human specification. The main problem isn't the AI — it's the human factor. SDD requires developers to specify their intentions precisely, which AI agents will ultimately execute. Yet this is exactly where the model faces its greatest challenge.

After over 10 years in software development, I have rarely experienced projects where requirements were completely formulated before implementation — neither by product developers nor by developers themselves. In my talks, I jokingly refer to us developers as "efficient," but "lazy" would probably be the more honest term.

Spec Kit encourages developers to continuously expand the generated documents and refine specifications. However, based on my examples from recent weeks, this approach doesn't automatically lead to better results. Only through the use of additional rules and MCP servers was I able to have my requirements — using GitHub Copilot — implemented cleanly and completely.

The question remains whether this model actually reduces development time or whether addressing a self-created problem — managing AI-generated code quality — leads to more frustration in the long term due to poorly optimized code architecture and increased specification overhead.

I will continue experimenting with Spec Driven Development in my projects and publish updates as the methodology and tooling mature. The early indicators suggest that while SDD adds upfront specification overhead, it may prove essential

for scaling AI-assisted development beyond prototype-level applications into production-ready systems.

The rapid evolution of Spec Kit — from v0.0.9 to v0.0.30+ in just weeks — demonstrates both the promise and volatility of this emerging methodology. As AI coding tools become more sophisticated and specification-driven development gains enterprise traction, we may be witnessing the birth of a new paradigm in software development.

Vibe Coding | AI | Coding | Spec Driven Development | Github

Follow

Written by Daniel Sogl

143 followers · 6 following

Software Engineer specializing in Angular, Ionic, & Capacitor. I build scalable apps, contribute to open source, and share knowledge via talks & articles.

Responses (2)

Bgerby

What are your thoughts?

Iones Walter
4 days ago

Excellent article! Thank you, Daniel!

As an enthusiastic product designer, I see SDD enabling rapid digital product prototyping and implementation. Cheers!

Reply

Nikos Katsikanis
Oct 2

Have you actually built anything with spec driven development and deployed to prod?

Reply

More from Daniel Sogl

Daniel Sogl

Analyzing and optimizing the esbuild Angular application bundle

Nov 15, 2024 104 1

Daniel Sogl

Gemini Code Assist for GitHub: Automated Code Reviews with Gemini

As a solo developer working on my side project codingrules.ai, I needed an efficient way to perform code reviews without another set of...

Apr 6 1 1

Daniel Sogl

How to upload coverage reports to SonarCloud using an nx monorepo

Merge your nx-project coverage reports into one file and upload them to SonarCloud using GitHub Actions.

Dec 3, 2022 23 3

Daniel Sogl

Functional Programming in Angular: Exploring inject and Resources

Angular’s evolving ecosystem is shifting toward a more functional and reactive programming paradigm. With tools like Signals, the Resource...

Dec 4, 2024 174 2

See all from Daniel Sogl

Recommended from Medium

ZIRU

Why Spec-Driven Development Made Me 5x More Productive

And How GitHub Spec Kit Changed Everything

Oct 16 28 2

Jettro Coenradie

Spec-driven development using Codex and Backlog.md

Don’t worry, this is not one of those blogs telling you we no longer need developers. In my daily work as a developer, I have become...

Oct 11 4

In Towards AI by Teja Kusireddy

We Spent $47,000 Running AI Agents in Production. Here's What Nobody Tells You About A2A and MCP.

Multi-agent systems are the future. Agent-to-Agent (A2A) communication and Anthropic's Model Context Protocol (MCP) are revolutionary. But...

Oct 16 1.6K 37

In AI Software Engineer by Joe Njenga

Cursor 2.0 Has Arrived — And Agentic AI Coding Just Got Wild

Cursor has released version 2.0 , bringing the most powerful agentic AI we have seen yet, more autonomous than ever before,here's what's...

2d ago 222 5

Dave Patten

Spec-Driven Development: Designing Before You Code (Again)

AI has changed how we build software - developers now code alongside assistants that analyze repos, write tests, and design architectures...

Oct 13 52 2

In Data Science Collective by Ida Silfverskiöld

Agentic AI: Single vs Multi-Agent Systems

Building with a structured data source in LangGraph

3d ago 378 9