Spec-driven development with AI coding agents: how we run it

By Andy Slack | October 31, 2026 | AI Development

What spec-driven development is, how it differs from vibe coding, and how we run it: who writes the spec, how it ties to tests, and who reviews the code.

Spec-driven development is a simple idea with a lot of noise around it. Before an AI coding agent writes any code, you write down what the software must do, clearly enough to test. Thoughtworks (https://www.thoughtworks.com/en-us/insights/blog/agile-engineering-practices/spec-driven-development-unpacking-2025-new-engineering-practices) describes it as using well-crafted requirement specifications as the prompts AI coding agents generate code from.

Most guides stop at the definition and a tool walkthrough. This one is about how we run it day to day: who writes the spec, what goes in it, how it is tied to tests and who reviews the result. For the loop the agents run once a spec exists, read The Agentic Development Loop (/articles/agentic-loop-development). This piece is about what goes in before it.

Spec-driven development vs vibe coding

With vibe coding you prompt an agent and keep what looks right. It is fast, and that is the problem. Thoughtworks calls vibe coding "too fast, spontaneous and haphazard", and points to the unmaintainable, one-off code it leaves behind.

Spec-driven development splits the work in two. First you agree what to build and how you will know it works. Then the agent builds it, and the tests prove it. Thoughtworks argues this is not waterfall again: it gives you shorter, more useful feedback loops, not longer ones. In our case each spec covers one change, not a whole project, so the loop stays short.

Where the spec sits, and who writes it

In our process, the spec comes before the loop. A person starts it as an issue in the repository: what is wanted, and a few acceptance criteria. A triage agent then reads the code and writes the full spec on that same issue.

No agent writes code until the spec exists, along with the failing tests it calls for.

What one of our specs looks like

One issue per change. It opens with what is wanted and three or four acceptance criteria. The triage agent then adds:

- The problem, as actual versus expected behaviour.
- The root cause, found by reading the code.
- A test plan in slices. Each slice is a failing test plus the change that makes it pass.
- The acceptance criteria, as a checklist.

- Each decision it made, with the reason.

Here is a real one. Our own issue to take two off-topic articles off this blog went from four criteria to five test slices, nine criteria and eight recorded decisions, and ended "Ready for tests: yes".

What we did before

This blog's engine was built from one long "master prompt": a role, an objective, every feature and a four-step plan, all in a single file. Once it had run, nothing tied that prompt to the code. It still asks for the Quill editor, but the site's editor is Tiptap.

Work now comes one small issue at a time, with acceptance criteria and failing tests that stay in the repository and run on every change. A prompt is read once; a test runs every time. Thoughtworks makes the same point from the other side: spec drift is hard to avoid, so you still need dependable automated checks to protect quality.

How the spec is tied to tests and review

Every change starts with failing tests. We work test-first, and the agents follow the same order every time:

- Tests first. One agent writes Playwright tests from the spec. They fail, because the feature does not exist yet.
- Code second. A separate agent writes the code. It can add test lines but not rewrite the ones it was given, so it cannot make a test pass by changing the test.
- Only the tests that matter. A command runs only the tests the branch adds or changes. It refuses to start if there are none, or if any test is skipped.
- Checks before a merge. Lint, the type check, the production build and every test must pass, and every page or route the branch changes must be reached by a test.
- Review. AI code review agents go through the work, with AI completeness checks and reviews for DRY and KISS code. If the application is public-facing or affects revenue, a person normally reviews the work in a staging environment too.

Does a clear spec change delivery?

Is there one project where a clear spec changed delivery? Every single one. That is the point of the spec: it takes business requirements and turns them into implementation requirements that an agent can build and a test can check.

Is it worth it for a small, fast-changing product?

Thoughtworks warns that over-formalised specs can slow down change and feedback, as early waterfall did. Ours start small: one issue per change, opening with three or four acceptance criteria.

So yes. We run all of our development through this process. It is quick and cheap, and it always saves more time in the long run. The more complex the system, the more time the spec deserves.

You keep the specs

Everything the agents work from lives in the repository, not in a tool of ours: the issues that hold the specs, the AGENTS.md file and the rules it points to, the tests, and the commands that run them.

No vendor lock-in: every solution we deliver comes with full training to self-administer. A client's team can keep writing specs and running the checks themselves, hire internally or move to another agency at any time.

Where to start

Take your next small change and write it as an issue: what is wanted, and three or four acceptance criteria you could test. If you use GitHub's open-source Spec Kit (https://github.com/github/spec-kit), which gives coding agents structured processes and templates, its workflow runs specify, plan, tasks and implement. Tests are optional there: its task template puts them first, written to fail before the code, only when the spec asks for them. Ask for them, and you get the order this article describes.

If you would like your team working this way, start with our AI adoption consulting (/consulting/ai-adoption), see how we handle bespoke development (/bespoke-development), or talk to us about consulting (/consulting). More on how we build with agents is in our articles (/articles).

Further reading

- Thoughtworks: Spec-driven development. Liu Shangqi on what a spec is, what makes a good one and whether this is waterfall again. Read the article (https://www.thoughtworks.com/en-us/insights/blog/agile-engineering-practices/spec-driven-development-unpacking-2025-new-engineering-practices)
- GitHub Spec Kit. The open-source toolkit, with its specify, plan, tasks and implement steps. See the repository (https://github.com/github/spec-kit)