

Spec-Driven Development

Could AI take teams from requirements to running code with more control?

...findings from product perspective | exploratory research

Darko Martić

WATERFALL

SEQUENTIAL ENGINEERING



AGILE

INCREMENTAL DELIVERY



AI BUILD

PROTOTYPE TO PLATEAU



VIBE CODING

NO PROCESS CHAOS



img src – https://lnkd.in/p/d448_he9



Could delivery be even better?

Classic delivery

Jira + manual dev

- 1 Keep requirements current during development
- 2 Catch misunderstandings before QA
- 3 Turn Figma designs into UI

AI coding as it is used today

"vibe coding"

- 1 Keep AI speed, make results reviewable
- 2 Capture decisions, not lose them in chats
- 3 Get consistent output across people

Is AI speed with requirement-level control possible?

How might SDD compare with today's approaches?

	Classic (Jira, manual dev)	Vibe Coding (AI helpers, no specs)	Spec-Driven Development
Source of truth	Stories go stale once dev starts	Decisions lost in chat prompts	One versioned spec that stays true
Requirements quality	Gaps surface in dev or QA	Gaps filled by AI guesses	Clarify and analyze catch gaps early
Change requests	Manual rework, story rarely updated	Quick edits nobody can trace	Edit the spec; plan, tasks, code follow
Consistency	Varies by developer	Varies by prompt	Enforced by constitution and spec
Figma → UI	Rebuilt by eye / partial automation	Partial automation	Agent reads frames and tokens
Traceability	Manual, partial	Effectively none	Requirement → task → test → code
Main cost	Slow delivery, rework	Hidden quality and audit debt	Upfront spec effort, new habits

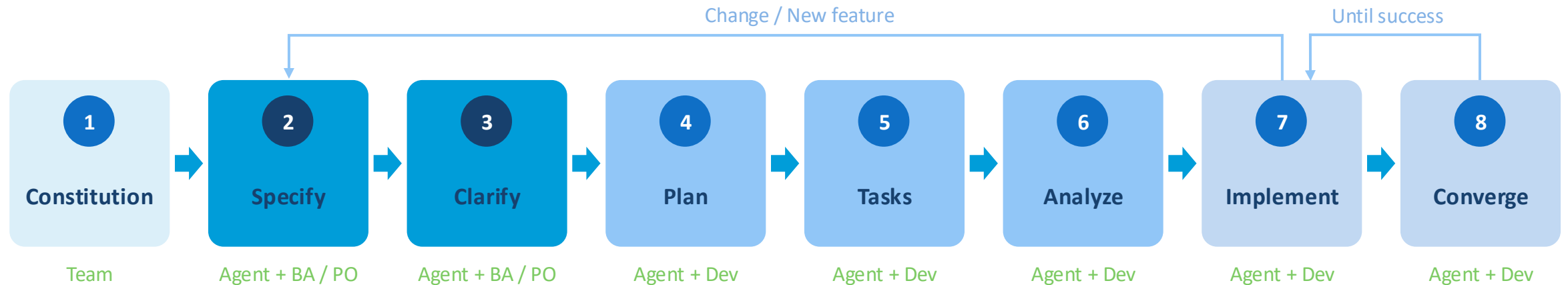
AI already makes code cheap; could SDD make it correct, consistent and traceable?

Upfront cost: a learning curve for every role, and change meets natural hesitation.
Teams that get time, a safe pilot and visible results move from scepticism to ownership.

One possible answer: spec-driven development

The spec, not the code, is the source of truth.

Change the spec first; the AI agent regenerates plan, tasks and code from it.



Change request

Edit spec.md first, then rerun clarify → plan → tasks → analyze → implement → converge.
Nobody edits code first, so spec, plan, tasks and code stay in agreement.

The artifacts: plain Markdown files in the repo

`todo-sdd/`

```
.claude/skills/  
  speckit-* skills  
.specify/  
  memory/constitution.md  
  templates/  
specs/  
  001-todo-list/  
    spec.md  
    plan.md  
    tasks.md  
index.html  app.js
```

`constitution.md`

Rules every feature obeys

Team

`spec.md`

WHAT and WHY: stories, FRs, acceptance

BA / PO

`plan.md`

HOW: tech approach, data model

Developer

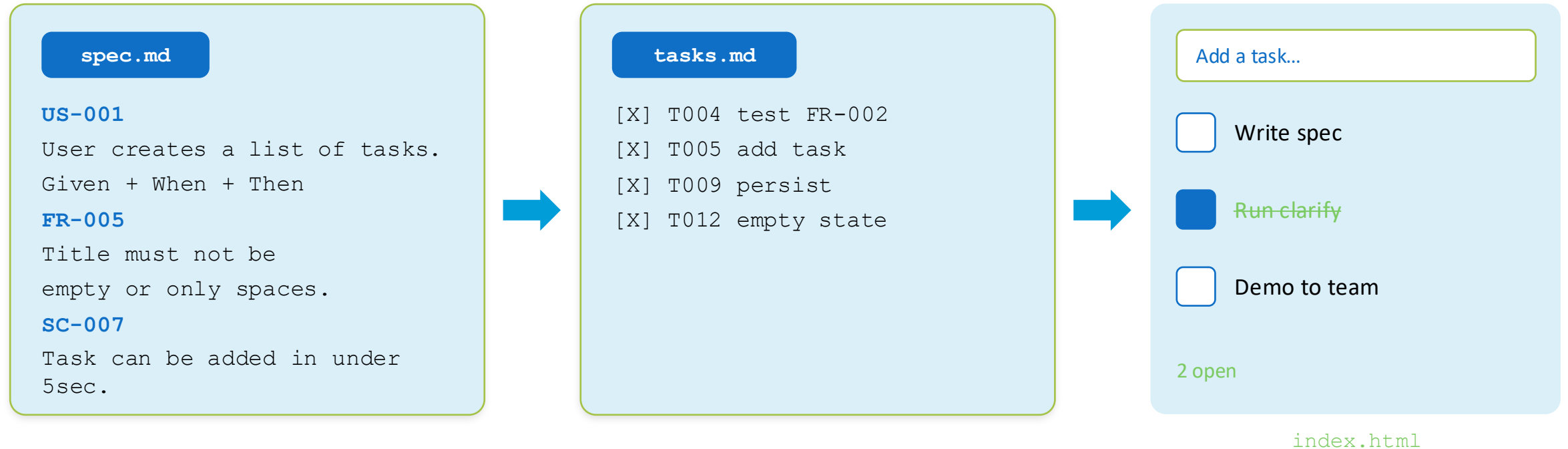
`tasks.md`

Ordered, checkable work items

Generated, dev reviews

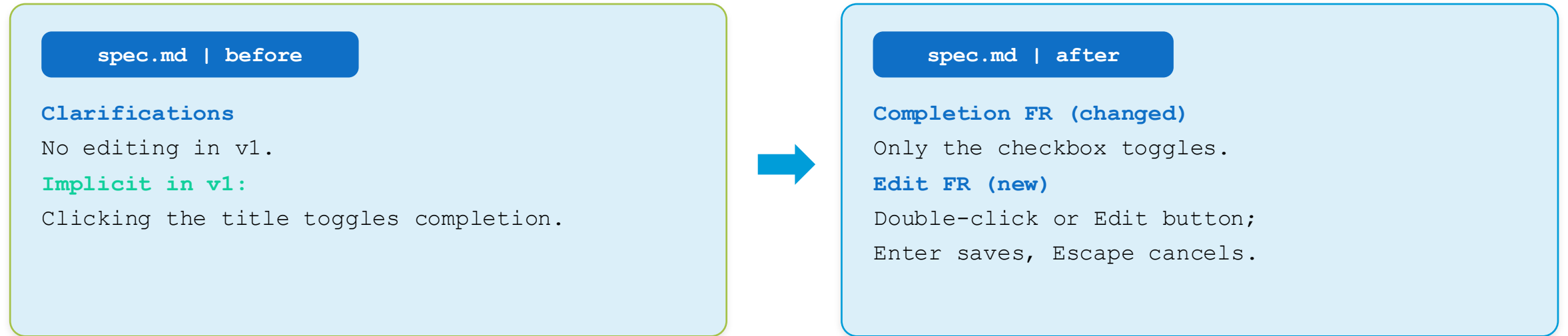
Demo: from spec to running app

Live demo | few min later



The agent built only what the spec asked for; every requirement maps to a test that runs in the browser.

Demo: a change request, "edit a task's title"



Note: adding the new story alone didn't work, because v1 let a click on the title toggle completion. A change can alter existing requirements, not just add new ones; the spec had to say so before the agent built it correctly.

How could roles work together?

BA / PO

Owns the spec

- Writes input, answers clarify
- Edits spec.md for every change
- Accepts against success criteria

Developer

Owns plan and build

- Plan, tasks, analyze
- Reviews every agent change
- Fixes bugs against FR IDs

QA

Owns verification

- Reviews spec for testability
- Tests from Given/When/Then
- Triage: code, spec, plan, design

UI designer

Owns look and layout

- Figma frames per user story
- design.md maps frames to FRs
- Visual review of the build

Jira

Tracks the work, links to the spec, never copies it

Figma

Agent reads frames via the Figma MCP

Sign-off

BA, dev, QA, designer approve before code

Could SDD support compliance needs?

What SDD could provide

- Documented requirements and change history
- Traceability: FR → task → test → code
- Constitution as written standards
- Analyze and converge reports as evidence

Controls to add

- Git history on a protected main branch
- Recorded approvals with name and date
- Segregation of duties: author ≠ approver
- AI governance: approved tools and data rules
- Automated tests with stored results
- Converge and tag before every release

PO sign-off = approval of an exact spec version, recorded in the repo and mirrored in Jira.

How could SDD scale to large projects?

Many small feature specs on a shared layer of project-wide context, never one big spec.

1 Feature-sized specs

One spec per vertical slice, 1–2 weeks, 2–6 user stories. Over ~10 clarify questions or ~40 tasks: split it.

2 Shared context layer

Constitution, glossary and domain model, architecture overview and API contracts, written once.

3 Dependencies & ownership

Specs declare dependencies; each product area has an owner; periodic cross-feature review.

4 Readable current picture

A living spec per capability; change specs feed into it.

5 Brownfield systems

Spec only the area being changed; the agent drafts current behaviour from code, the BA corrects it.

6 Phased implementation

Implement big features phase by phase and converge after each one.

Keep an eye on: spec sprawl, cross-feature consistency, and the upfront effort of the shared layer.

When could SDD be worth adopting?

Good fit	Poor fit
Product maintained for years	Throwaway prototypes, spikes, exploratory R&D
BA, dev, QA (and designer) need one shared truth	Tiny fixes, config changes, one-liners
Rework caused by misunderstood requirements	Requirements genuinely unknown, stakeholders unavailable
Devs already use AI agents and output is inconsistent	Mid-crunch on a critical release
Traceability or audit needs	No AI tool approval in place
Handovers, new hire onboarding	

Readiness before a pilot

AI tool approved by security and compliance | basic repo practices | a tech lead owning the constitution | BAs ready to write precise specs | devs reviewing all agent output | acceptance of a slower start.

Questions to decide together

Possible next steps

1

Is a small pilot worth running, once the readiness checklist is met?

2

Which framework and model fits the pilot team?

3

Does a phased rollout make sense?
pilot → team default → org standard