SPEC-DRIVEN DEVELOPMENT (SDD) · THE WORKFLOW

The spec is the new source code: how spec-driven development works

When AI agents write most of the code, the scarce skill stops being typing and becomes saying precisely what you want. Spec-driven development turns that statement into versioned files — a constitution, a spec, a plan, a task list — that an agent executes and a human reviews. GitHub, AWS, and JetBrains shipped tooling around it within a year, and the reference toolkit has passed 124,000 GitHub stars. Whether SDD is the new discipline of software or waterfall in a new jacket is the fight this brief takes seriously.

AUTHOR

AT

Alicante Tech Vibes *Research Labs*

~10 MIN READ · ALL FIGURES SOURCED & DATED

124k+ GITHUB STARS ON SPEC KIT, THE OPEN-SOURCE SDD REFERENCE TOOLKIT (AUG 2026)	**4 files** CONSTITUTION · SPEC · PLAN · TASKS — THE ARTIFACTS EVERY SDD RUN PRODUCES	**30+** AI CODING AGENTS SPEC KIT PLUGS INTO (COPILOT, CLAUDE CODE, GEMINI CLI…)
90% OF DEVELOPERS NOW USE AI AT WORK — MEDIAN 2 H/DAY (DORA, SEP 2025)	**24%** REPORT HIGH TRUST IN AI-GENERATED CODE — THE GAP SDD EXISTS TO CLOSE (DORA)	**$20–200** MONTHLY PRICING OF AWS KIRO, THE FIRST SPEC-NATIVE IDE (GA NOV 17, 2025)

I. From vibes to contract

For two years the dominant way to code with AI was what the industry now derides as vibe coding: describe the goal in a chat box, accept whatever appears, iterate until it compiles. Spec-driven development is the correction. As Pachi Parra's practical introduction puts it, SDD starts from a written specification — what the system should do, how it behaves in every scenario, which constraints it must obey — and treats the AI agent as the executor of that contract rather than a slot machine. The idea is old; the tooling wave is new. GitHub open-sourced Spec Kit on September 2, 2025, AWS took Kiro — an entire IDE built around specs — to general availability on November 17, 2025, and JetBrains co-produced a DeepLearning.AI course on the method. **In under a year, "write the spec first" went from blog-post advice to the default workflow shipped by two of the industry's largest platform vendors.**

The reason is mechanical, not fashionable. GitHub's engineers describe coding agents as "literal-minded pair programmers": given a vague prompt, a model guesses at thousands of unstated requirements, and every guess is a place the result can silently diverge from intent. A spec converts guesses into decisions made once, in writing, where a human can check them.

II. The four files

SDD's entire mechanism is a chain of Markdown artifacts, each generated by the agent from the previous one and each gated by human review. In Spec Kit's canonical form, slash commands (`/speckit.constitution`, `/`

`speckit.specify`, `/speckit.plan`, `/speckit.tasks`, `/speckit.implement`) produce four files:

`constitution.md` — the project's non-negotiables.

Coding standards, architectural principles, security rules, testing requirements. Written once, enforced on every feature; this is what keeps fifty agent runs stylistically coherent.

`spec.md` — the what and the why.

User journeys, behavior in every scenario, edge cases, success criteria — deliberately technology-agnostic. This is the file product owners can actually read and correct.

`plan.md` — the how.

Stack choices, architecture, data model, API contracts, constraints. The same spec can legitimately produce different plans; the plan is where engineering judgment lives.

`tasks.md` — the work, atomized.

Small, independently testable work items derived from the plan, executed one by one (Kiro runs independent tasks in parallel "waves") with status tracked in the file itself.

AWS Kiro implements the same chain as requirements.md, design.md, and tasks.md, writing acceptance criteria in EARS notation (a controlled natural language for requirements inherited from systems engineering), plus persistent "steering" files for team conventions. Optional passes — `/speckit.clarify` to interrogate ambiguities, `/speckit.analyze` to check the artifacts against each other, `/speckit.converge` to detect drift between spec and codebase — round out what practitioners describe as a seven-phase loop: constitution → specify → clarify → plan → tasks → implement → analyze.

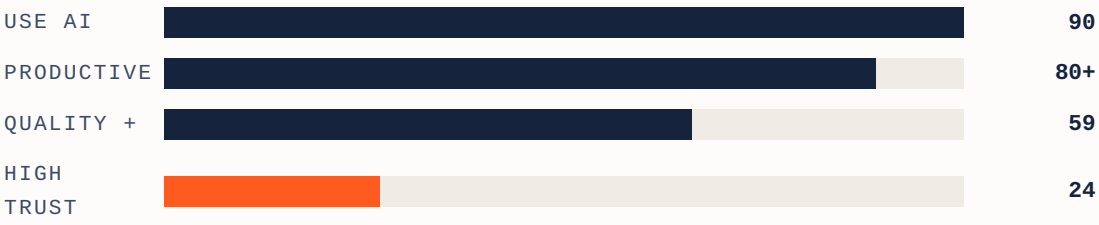
> ***“The spec is now where the thinking happens.”***

FROM A PRACTITIONER SURVEY OF HOW TEAMS ACTUALLY USE SDD IN 2026 — DEV.TO (KRLZ), 2026. CODE REVIEW BECOMES SPEC REVIEW; THE DIFF THAT MATTERS IS THE ONE IN SPEC.MD.

III. Why now: the trust gap

The adoption math explains the timing. Google's 2025 DORA report (≈5,000 professionals, September 2025) found 90% of developers using AI, a median of two hours a day working with it, and over 80% reporting productivity gains — while only 24% expressed high trust in AI-generated code. Stack Overflow's 2025 survey found the same shape: usage at 84% and rising, trust at an all-time low. That combination — everyone uses it, nobody trusts it — is precisely the condition a verification-forward workflow resolves. SDD moves human attention from the least reviewable artifact (ten thousand lines of generated diff) to the most reviewable one (a page of declared behavior), and the agent's compliance with tasks.md is checked by tests, not by eyeballing.

FIG. 1: EVERYONE USES IT, NOBODY TRUSTS IT — THE GAP SDD TARGETS (% OF DEVELOPERS, DORA 2025)



BARS SCALED TO 90% = 100%. SOURCE: GOOGLE CLOUD, 2025 DORA / STATE OF AI-ASSISTED SOFTWARE DEVELOPMENT, SEP 23, 2025, N≈5,000.

IV. What actually changed about coding

In an SDD shop the developer's day inverts. The morning's work is writing and reviewing prose: sharpening the spec, answering the agent's `/clarify` questions, editing the plan's architecture before a line exists. Implementation — formerly the job — becomes a supervised batch process. Pull requests increasingly lead with the spec diff, because that is where a reviewer can still catch a wrong decision cheaply; by the time it's code, the error is expensive and camouflaged. The senior skill shifts from knowing *how* to build to specifying *what* and constraining *how* — and the constitution file quietly becomes the most

leveraged document in the repository, since every future agent run inherits it. GitHub's own guidance says the method pays off in three settings: greenfield builds, feature work in existing systems, and legacy modernization — which is to say, most professional software work, and a natural companion to labs' own disclosure that AI already authors the majority of code at the frontier.

v. Standard, or ceremony?

Calling SDD "the new industry standard" outruns the published evidence, and the honest read says so. The momentum is real: a 124k-star reference toolkit, a spec-native IDE from AWS, native workflows in Claude Code and Cursor, and a crowded second ring (OpenSpec, Tessl) mapped by practitioners. But the same survey warns that quantified benefits — higher first-pass agent success, error reductions — are vendor-reported and **"directional, not proven."** The skeptic case has four teeth: SDD can be waterfall repackaged, front-loading decisions that exploratory work can't make yet; specs drift from code the moment pressure rises, and `/converge` is a mitigation, not a law of physics; over-specified specs degenerate into pseudo-code, re-creating the work they were meant to replace; and a perfectly implemented wrong spec yields confident failure. Practitioners' own guidance is to skip SDD for throwaway prototypes and genuinely exploratory work — a scoping the loudest advocacy omits.

vi. The toolbox: repositories and tools worth starring

Six tools define the SDD landscape in mid-2026. Three are open-source repositories; three are commercial products. The one hands-on comparison across all of them — Augment Code's tested roundup (greenfield API, brownfield Express.js feature, four-service refactor) — is useful but sells its own entry, so its comparative verdicts are read here as vendor claims. The structural split it identifies is real, though: **static specs** (written once, drift as code evolves) versus **living specs** (updated as agents implement).

`github/spec-kit` — the reference implementation. github.com/github/spec-kit

MIT license, 124k+ stars, works with 30+ agents (Copilot, Claude Code, Gemini CLI, Cursor, Windsurf). The full constitution → specify → plan → tasks → implement chain, installed via `specify-cli`. Maximum portability, zero lock-in; specs are static, and budget 1–3 h per feature including review.

`Fission-AI/OpenSpec` — lightest for existing codebases. github.com/Fission-AI/OpenSpec

MIT, 52k+ stars, the most actively maintained open-source SDD framework. A strict proposal → apply → archive state machine: `openspec/specs/` holds current truth, `changes/` holds proposals with delta specs, and validation catches missing acceptance scenarios before code is written. Built brownfield-first — the right default for iterating on running systems.

`bmad-code-org/bmad-method` — the full agile simulation. github.com/bmad-code-org/bmad-method

MIT, 51k+ stars. Orchestrates 12+ named agent personas (PM, Architect, Developer, QA…) through 34+ workflows, generating PRDs, architecture docs, and granular stories. The most thorough documentation machine in the set; the persona handoffs add coordination overhead that punishes rapid iteration and small teams.

Amazon Kiro — the spec-native IDE. kiro.dev

The requirements.md / design.md / tasks.md chain with acceptance criteria in EARS notation (“WHEN [condition] THE SYSTEM SHALL [behavior]”), agent hooks, and a 2026 requirements-analysis feature that uses SMT solvers to catch contradictory requirements before generation — its genuinely differentiated capability. Free tier; Pro $20, Pro+ $40, Power $200/month. Specs are static, and it's a dedicated IDE: ecosystem lock-in is the price.

Augment Cosmos — the organizational layer. augmentcode.com

Launched May 2026: living specs that update as implementing agents change interfaces, a context engine the vendor says spans 400k+ files, cross-session organizational memory, and event triggers from GitHub, Linear, Slack, and PagerDuty. Business plan $100/month flat. The living-spec model directly targets the drift problem — but the claims are vendor-reported with few independent benchmarks yet, and the scope is overkill below roughly 20 engineers.

Cursor rules — the minimum viable spec. cursor.com/docs

Not full SDD: `.cursor/rules/*.mdc` files act as persistent, glob-scoped system prompts encoding conventions and architecture decisions. Zero migration cost if the team already lives in Cursor ($20–200/month tiers), but there's no spec lifecycle, no validation, and rule activation gets inconsistent as files multiply.

Choosing between them reduces to two questions. Does the work live in one repo or across services? — single-repo teams are served by Spec Kit or OpenSpec for free; multi-service coordination is where static Markdown breaks and platform-scope tools argue their fee. And do requirements hold still? — stable, well-understood domains suit static specs (Kiro, Spec Kit); evolving multi-session work either needs living specs or the discipline to re-run `/speckit.converge` religiously.

VII. How it plays out: three scenarios

Horizon 2026–2029. Probabilities are analytical judgment, not measurement.

1 · The default discipline ~50%

2026–2029 · STEADY

SDD becomes for agentic coding what code review and CI became for the last era: assumed for professional team work, skipped for scratch projects, embedded in every serious tool. Watch: spec artifacts appearing in public repos at scale; enterprise platform teams mandating constitution files; the 2026 DORA report measuring spec-first workflows explicitly.

2 · Absorbed and invisible ~30%

2027–2029 · QUIET

The practice wins but the ritual disappears: agents generate, maintain, and reconcile specs internally, surfacing them only when a human needs to arbitrate. The four files become an implementation detail of the IDE, the way compilers made assembly invisible. Watch: Kiro/Copilot auto-generating specs from conversation without named phases; spec maintenance offloaded to background agents; "SDD" fading as a term while its artifacts persist.

3 · Ceremony collapse ~20%

2026–2028 · REVERSAL

Models get reliable enough at inferring intent that the overhead stops paying: teams measure spec-writing time exceeding rework saved, drift makes the artifacts actively misleading, and SDD joins the graveyard of heavyweight methodologies. Watch: independent studies showing no defect/throughput advantage for spec-first teams; Spec Kit contribution activity declining; vendors quietly de-emphasizing the workflow in favor of longer-autonomy agents.

WHAT WOULD FALSIFY THIS THESIS

The thesis: SDD is the leading candidate for the standard discipline of agent-written software, because it relocates human review to the highest-leverage artifact. It fails outward if rigorous independent measurement — a DORA-class study, not vendor benchmarks — finds spec-first teams shipping no better than prompt-first teams; today the quantified upside is explicitly "directional, not proven." It fails inward if agent intent-inference improves so fast that written specs become redundant overhead — the same capability trend (METR's ~89-day doubling of autonomous task horizons) that makes SDD necessary today could make it obsolete. The strongest current counter-case is the drift problem: code, not the spec, remains the operational source of truth in every deployed system.

WHAT TO WATCH, IN ORDER

1. **DORA 2026 report** (cloud.google.com, expected ~Sep 2026): whether spec-first workflows get measured as a category, and whether they correlate with delivery performance.
2. **Spec Kit repository trajectory** (github.com/github/spec-kit, ongoing): stars, release cadence, and whether `/converge`-style drift tooling matures — the practice lives or dies on maintenance cost.
3. **Kiro adoption signals** (AWS announcements, re:Invent Dec 2026): team-tier uptake and case studies beyond AWS's own blogs.
4. **Spec artifacts in the wild** (GitHub search, quarterly): growth of `constitution.md` / `spec.md` files in public repositories — the cleanest non-vendor adoption proxy.
5. **Independent effectiveness studies** (arXiv / METR-style RCTs, 2026–2027): any randomized comparison of spec-first vs. prompt-first agent development; none exists yet.

SOURCES

1. Pachi Parra, “A Practical Intro to Spec-Driven Development (SDD),” DEV Community, June 2026 · dev.to
2. GitHub, “Spec-driven development with AI: Get started with a new open source toolkit,” September 2, 2025 · github.blog
3. github/spec-kit repository (stars, commands, artifacts), accessed August 12, 2026 · github.com
4. Kiro documentation, “Specs” (requirements.md, design.md, tasks.md), accessed August 12, 2026 · kiro.dev
5. SiliconANGLE, “AWS launches Kiro into general availability,” November 17, 2025 · siliconangle.com
6. Google Cloud, “2025 DORA Report / State of AI-assisted Software Development,” September 23, 2025 · blog.google
7. Stack Overflow, “2025 Developer Survey” press release, 2025 · stackoverflow.co
8. krlz, “Spec-Driven Development in 2026: What It Is, the Tooling, and How Teams Actually Use It,” DEV Community, 2026 · dev.to
9. Anthropic Institute, “When AI builds itself,” June 5, 2026, via The Next Web · thenextweb.com
10. METR, “Time Horizon 1.1,” January 29, 2026 · metr.org

11. Augment Code, “6 Best Spec-Driven Development Tools for AI Coding in 2026” (hands-on comparison; vendor of Cosmos), 2026 · augmentcode.com
12. Fission-AI/OpenSpec repository (stars, workflow), accessed August 12, 2026 · github.com
13. bmad-code-org/bmad-method repository (stars, personas, workflows), accessed August 12, 2026 · github.com

METHOD NOTE: FIGURES ARE ATTRIBUTED AND DATED INLINE; VENDOR-REPORTED EFFECTIVENESS CLAIMS ARE LABELED AS SUCH; WHERE INDEPENDENT SOURCES DISAGREE, THE DISAGREEMENT IS REPORTED RATHER THAN AVERAGED. SCENARIO PROBABILITIES ARE ANALYTICAL JUDGMENT.

Built by the Alicante Tech Vibes community, a tope. 2026-08-12