For Engineering Leadership

Spec-Driven Development

Structure for the work. Proof for the claims.

Prepared by **Grant Howe, Managing Partner**

Edition 2 · August 2026 · describes methodology v4.0.5

Geekbyte

PE Portfolio Technology Leadership

Practitioner Brief — for engineering leadership Prepared by Grant Howe, Managing Partner, Geekbyte LLC Edition 2 · August 2026 · describes methodology v4.0.5

The Gap Every Engineering Leader Is Explaining

Your developers are using AI coding tools, and the tools are working. They are faster, and by 2025 the organizational throughput number had turned positive too. What did not follow is stability. More is reaching production, and more of what reaches production breaks.

84%	+98%	-7.2%	Reversed
of developers use or plan to use AI tools	pull requests merged per developer	delivery stability, per 25% more AI adoption (2024; still negative in 2025)	throughput fell 1.5% in 2024, rose by 2025 — stability did not follow

Sources: 2024 and 2025 DORA State of AI-assisted Software Development reports; Faros AI study of 10,000 developers; 2025 Stack Overflow Developer Survey.

Faros named the volume half of it the AI Productivity Paradox. Individual engineers really are faster at the task in front of them. What has not recovered is everything downstream of writing the code — and that is because AI changed something more fundamental than speed.

When humans wrote every line, the writing was the evidence. Slow, expensive code carried an implicit warranty: someone thought about this. AI removed the cost of producing code and, with it, the cost of producing claims. "It works" is now cheap to say and expensive to verify. Review queues built for human volume are absorbing machine volume, reviewers approve what they cannot fully absorb, and the gap between what is asserted and what is proven widens with every sprint.

Spec-Driven Development is built for that problem. It is a methodology for running an engineering organization where AI does much of the writing — structure for the work, and proof for the claims.

The Structure Layer: Specs, Tiers, One Operator or Many

SDD starts from a rule that predates AI and matters more because of it: work does not begin until what is being built, and why, is written down.

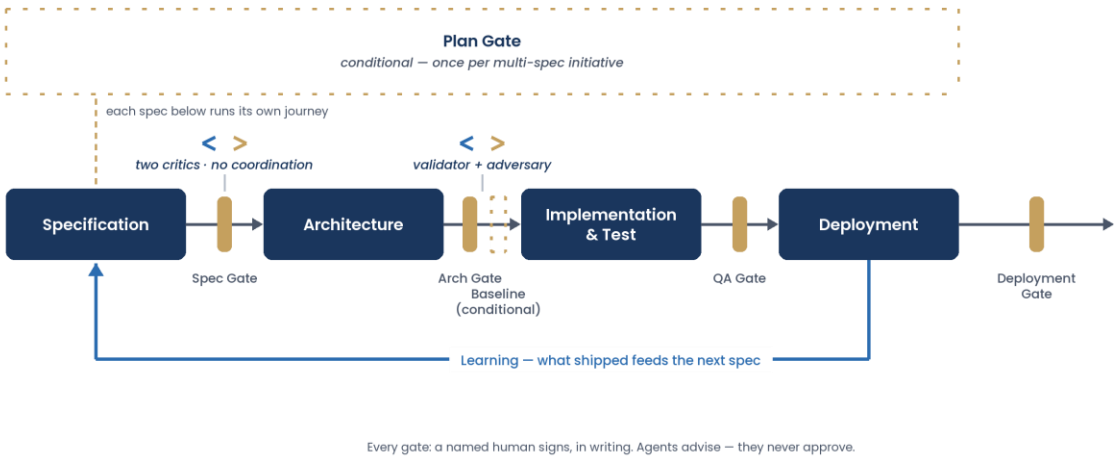
Two spec types. *Feature Specs* describe what to change — the intent, acceptance criteria, and constraints for a unit of work. *Anchor Specs* describe how the system is — the durable architectural truths new work must not silently violate. Feature Specs are consumed by the pipeline; Anchor Specs are consulted by it. Together they are the written record that review, onboarding, and — later — diligence draw on.

Complexity tiers. Not every change deserves the same ceremony. A copy edit and a payments change do not take the same path. Every spec declares a tier, the tier sets the depth of review it must survive, and the escalation triggers that force a higher tier live in one canonical governance file — so a trigger cannot read one way in one document and another way in a second.

The Solo Operator Model. SDD is written to run at both ends of the scale: a full team with named owners per seat, or a single senior operator running every seat with the role separation preserved in the record. The gates do not thin out when the team does — what changes is who fills them, not whether they are filled.

The Pipeline and Its Gates

Work moves through a staged pipeline — specification, architecture, implementation and test, deployment, and a learning stage that feeds what shipped back into how the next spec is written. Between stages sit gates, and the gates are the point: work does not advance until a named human signs off, in writing, with the evidence the gate requires attached.



Every gate: a named human signs, in writing. Agents advise — they never approve.

When one objective is too big for a single spec, a *Plan Gate* approves how it breaks into ordered phases before any of it is built. A standalone feature never needs one. It sits at a different altitude from the pipeline above — one Plan Gate over a body of work, not a planning step inside a spec. A conditional *Baseline Gate* sits at spec altitude, between the Architecture Gate and implementation, and fires only when a spec declares it will touch existing behavior — capturing the before-state before implementation destroys it, so parity can be proven element by element rather than remembered.

One rule governs every gate: **AI agents advise; they never self-approve.** The pipeline's agents assemble evidence, argue positions, and surface divergence. A human owns every approval, and the approval is an artifact — which is what makes the record worth something later.

No Unopposed Approvals

The most common failure mode of review — human or AI — is agreement. A single confirmatory reviewer reinforces the author's priors; a reviewer who shares the author's context finds the author's conclusions. SDD's answer is structural: at the two gates where judgment matters most, review is opposed by design.

At the Architecture Gate, two architects. One validates the design. A second — the adversary — hunts the riskiest assumption and the failure mode the validator did not look for. Neither sees the other's output first. Where they diverge, the divergence is

presented as the finding, never reconciled into one comfortable view. The disagreement is the signal.

At the Spec Gate, antagonistic review. For complex and critical work, two critics with different lenses and no coordination read the spec before it is approved — one attacking the substance, one reading it cold from the recipient's chair. Verdicts are Pass, Pass with conditions, or Fail, and every finding is graded with the same proof vocabulary the rest of the methodology uses.

This is not process for its own sake. It is the cheapest known defense against the confident, fluent, wrong answer — from a model or from a person.

Proof Discipline

The vocabulary underneath every review in SDD is a three-tag system:

PROVEN — observed in the real running system. **HYPOTHESIS** — reasoned, plausible, not yet verified. **RETRACTED** — asserted once, now withdrawn, left on the record.

The rules around the tags do the work. Proxies — static reading, logs, unit tests, a model's own reasoning — can *falsify* a claim but never *confirm* one; only the running system confirms. No probe starts without a kill criterion: the observation that would prove the investigator wrong, named before the investigation. Findings are attacked by their own author before they are presented. Settled decisions are not reopened without new evidence.

> ***"Not yet proven" is a correct answer, not a failure to have one.***

For an engineering leader, the tags change two conversations. Status stops being vibes — "done" decomposes into what is proven, what is hypothesized, and what was retracted, and the mix is visible. And AI output gets a handling class: a model's claim about the code enters as HYPOTHESIS by definition, and something real has to happen before it graduates.

Critical by Default

Some changes do not get to argue about their tier. Authentication and authorization, payment and financial data, personal data, and changes to the core domain model

are **Critical by default — and the default itself is immovable.** What the operator controls is the override, and the override is recorded, not verbal: classifying a spec below Critical requires a written rationale carried in the spec itself, and the external second reviewer that Critical tier expects can be declined by a solo operator exactly once per spec, at authoring time, with the declination recorded and carried forward by every gate that follows.

The design intent is auditability without bureaucracy. The rule never moves; the exceptions are cheap to make and impossible to make silently. Eighteen months later, every case where the high-risk path was not taken has a name, a date, and a reason attached.

Every Criterion Measured, Every Assumption Killable

The spec template enforces two disciplines at authoring time that most organizations bolt on after the fact, if ever.

Measurement per criterion. Each acceptance criterion names the tool that measures it, the threshold that constitutes passing, and the evidence artifact attached at sign-off. "Works" is not a criterion; a named measurement with a named threshold is.

Kill criteria. Every spec states the assumptions it is betting on, the single observation that would kill each one, and what fires if that observation arrives. Specs stop being bets nobody remembers making.

The Working System

SDD ships as a working kit, not a binder. The methodology, the governance files, the gate checklists, and the spec templates install into a repository alongside two agent packs that run the pipeline inside Claude Code: the five pipeline agents that move work stage to stage, and the review and authoring seats — including the adversary architect and both spec critics — that staff the opposed reviews. A two-seat security review is wired into the architecture gate: one seat finds and traces candidate weaknesses, a second attacks what survived.

Three optional layers extend it — hooks that catch silent failures, skills that encode required methods, and commands that make the never-self-approve rule executable — each added by adding a directory, not by editing an installer. The installer itself is built for adopters who customize: it fingerprints what it delivered, never clobbers what you changed, and respects what you deleted.

A coach installs alongside the kit and walks a team through adoption incrementally — the methodology maps onto the team that exists, whether that is eight engineers or eighty, without a rewrite, a re-platform, or a reorganization.

Evidence From Use

31

adjudicated fixes in the v0.1.8 hardening pass, with a second pass behind v0.1.9

The kit is hardened the way it says software should be: by adversarial use, on the record.

It has taken and answered a real adopter defect report from a clean-project install — four defects, one a blocker, each fixed with a regression check added so the class of failure cannot ship again. A full hardening pass ran over the shipped corpus and its thirty-one adjudicated fixes landed in v0.1.8; a second pass ran over the shipped kit for the current release. Its shipped documents are provably generated from tracked sources. And it carries its first outside-principal contribution — the methodology is developing a contributor base beyond its author.

That is the honest state of the evidence: not a promise that the methodology is finished, but proof that it is alive, in use, and improving under the same proof discipline it prescribes.

Adopting It

The methodology is open source under AGPL-3.0. Any team can read it, install it, and run it without us — that is deliberate, and the documents your team would evaluate are the same ones this brief describes.

Geekbyte's role in a portfolio engagement is the operator alongside the CTO while it lands: what to formalize first, how to sequence adoption against a committed roadmap, where the tiers should sit for this team's risk profile. My background is 25+ years as a CTO and SVP of R&D, 27+ M&A integrations, and four PE exits — the judgment calls above are where these adoptions succeed or stall, and they sit in operator experience, not in a document.

The next move: thirty minutes with your engineering leadership. We walk the pipeline against one of your real specs and you will know by the end whether this fits how your team should work.

Grant Howe, Managing Partner grant@geekbyte.biz · geekbyte.biz/methodology