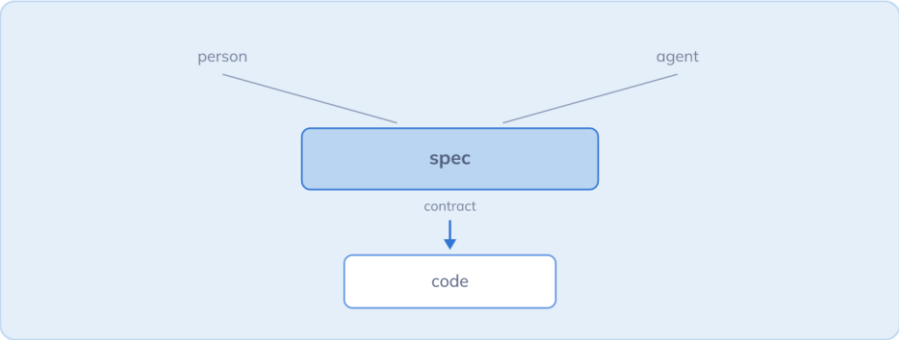
Scrum Manager
Uncovering better ways

## State of the art

# Spec Driven Development

## in agile teams

Reference map of current professional knowledge




**Updated September 2026**

**Scrum Manager® · Skill Arena**

## About this document

This document is a curated map of the knowledge, practices and models in professional use as of its publication date for steering AI-assisted software development through specifications (Spec Driven Development) in agile teams. For each entry, the map places it in context, indicates its degree of adoption in current professional practice and points to where to find reference information.

Use it as an observatory and reference point to check whether the professional knowledge you rely on is aligned with current practice and at the leading edge.

It is complemented by an in-depth companion guide that develops each concept and by the training and assessment platform in Skill Arena. In the Spec Driven Development in agile teams area you can test your level of knowledge and, if you pass, obtain a diploma that provides curricular accreditation of professional competence and currency in this area.


## State of knowledge

Knowledge about Spec Driven Development evolves extremely fast: method, tools, and practices are renewed within months. The fundamentals are already settled and the method gained academic and institutional backing during 2026, but the toolkit is still taking shape and the evidence on how much AI really speeds up development is under revision. Throughout the document, the labels (ESTABLISHED, CONSOLIDATING, EMERGING) help identify the maturity of each concept:

**ESTABLISHED** settled consensus; knowledge taken as required.
**CONSOLIDATING** gaining adoption fast; not yet universal but already relevant.
**EMERGING** recent frontier; high relevance and high volatility.

# Block A — The paradigm: why SDD

*What SDD is, what problem it solves, and why it is not a step back to waterfall. It is the conceptual base common to every role on the team.*

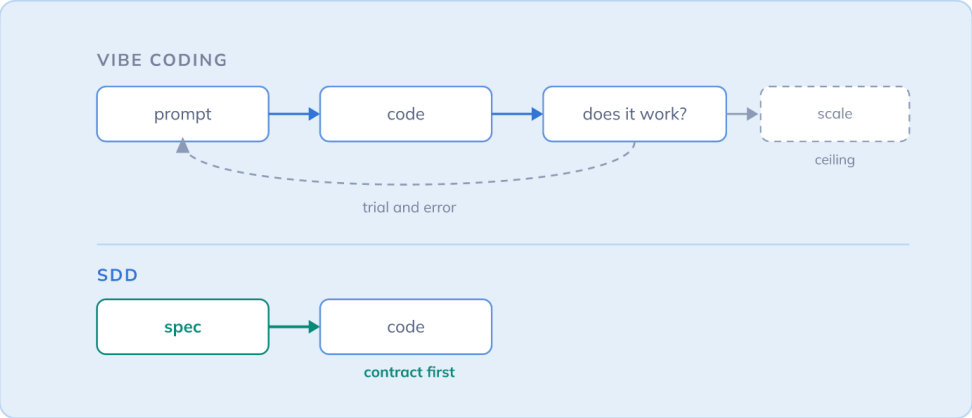



***Figure 1.*** *SDD inverts the order: vibe coding converses with the agent and SDD hands it a contract before generating code.*

## From vibe coding to spec-driven development · ESTABLISHED

Vibe coding consists of asking the AI for code conversationally and accepting it if it works. It is extraordinarily productive for prototypes, internal tools, and exploring ideas, but it has a ceiling that appears sooner than most people expect: beyond a certain scale it generates implicit requirements, context contamination, and architectural drift. Spec-driven development answers by inverting the order, with a spec that acts as a contract before a single line of code is generated.

**Why it is here now.** The discussion has matured and so has the evidence, which should be handled with care. METR's controlled trial of July 2025, with 19% more time spent with AI against a perceived 20% speed-up, is still the most cited figure in the field; its own author redesigned it in February 2026, on finding that the tracking could no longer distinguish the effect from its absence, and today believes the real speed-up is probably greater. What does hold, and what the DORA report on the return on investment in AI describes by name, is the verification tax: the effort of checking that generated code is reliable and consistent with the architecture does not disappear, it moves.

**Where to look.** METR — change to the experiment design (2026) · DORA — ROI of AI-assisted Software Development

## The spec as the primary artifact of development · ESTABLISHED

The center of the work shifts from writing code to specifying clearly what is built, why, under what constraints, and with what success criteria. The spec is the blueprint and the code is the consequence derived from it. For a literal agent, every decision the spec does not make is a decision the agent makes on its own.

**Why it is here now.** It is the change of professional identity that runs through the topic, and it affects every role on the team and not only the programmer, because specifying

precisely is a shared competence. During 2026 the idea has gained academic formulation: the work of Díaz, Gayoso, Cimmino, and Pérez describes specifications as the contractual substrate between humans and agents, and holds that it is what restores the accountability and verifiability that informal AI assistance erodes.

**Where to look.** Addy Osmani — How to write a good spec for AI agents · SDD for Agentic Software Engineering (arXiv, 2026)

### SDD and agility: not the return of waterfall · CONSOLIDATING

SDD uses sequential phases and approval gates, which is reminiscent of waterfall, but it differs on three axes. The scope is each increment and not the whole project, feedback arrives at every gate and not at the end, and reversibility is high because changing a document is trivial compared with rewriting code. Properly understood, it is the coherent application of agile principles when part of the team is agents.

**Why it is here now.** It is the objection any agile professional raises when faced with SDD, and resolving it is what allows adopting it without betraying agility. The critique has names and is still alive: Zaninotto maintains that SDD revives the heavy documentation that precedes code and proposes natural-language development as the alternative, and Yeret has debated from the agile world whether it is a step forward or back. The Thoughtworks Technology Radar keeps it in “Assess”, without having promoted it to “Trial”: it adds value with ambiguous requirements, distributed teams, or a need for traceability, and it may not pay off for trivial problems.

**Where to look.** Thoughtworks Technology Radar — SDD · Marmelab — SDD: The Waterfall Strikes Back

## The principle of proportionality · ESTABLISHED

It consists of adjusting the intensity of the process to the size of the problem. A trivial change does not need a four-phase spec; a complex feature does. Badly calibrated, SDD becomes a specification bureaucracy that slows down instead of speeding up.

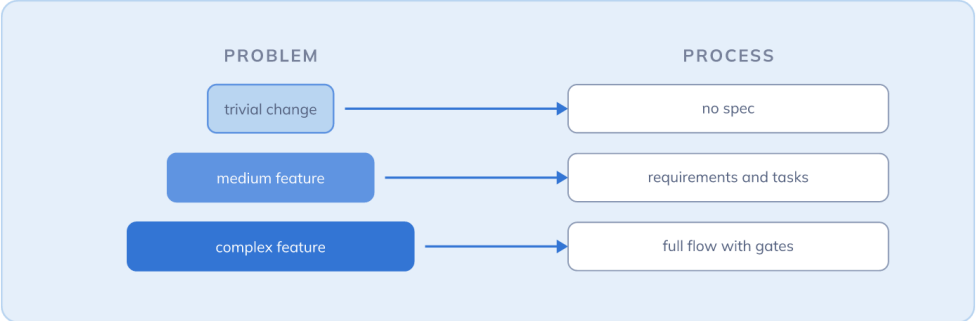



***Figure 2.*** *The principle of proportionality: the intensity of the process is adjusted to the size of the problem.*

**Why it is here now.** It is the criterion that separates the useful use of SDD from its bureaucratic caricature. Böckeler documented how a tool turned a minor bug into four stories with sixteen acceptance criteria, and described it as using a sledgehammer to crack a nut. The evidence accumulated during 2026 reinforces the principle from another angle: the return on AI is high in new, simple work and much lower in legacy, complex code, so the intensity of the process cannot be the same in both cases. Knowing when not to apply SDD is part of mastering it.

**Where to look.** Böckeler — Understanding Spec-Driven Development · Scrum Manager — Scrum in the age of AI (in Spanish)

# Block B — The method: phases, wrappers, and gates

*The mechanics of SDD: the workflow, the artifacts that frame it, and the approval gates that make it viable. It is the operational core of the topic.*

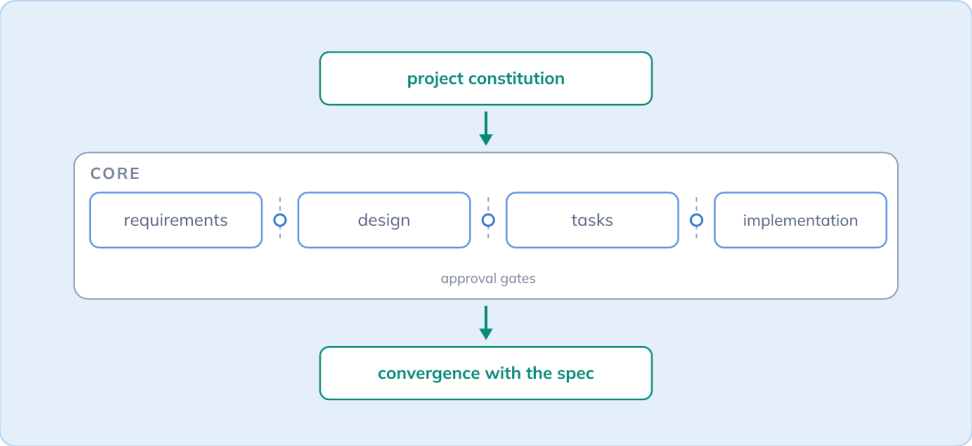



***Figure 3.*** *The four-phase core and its two wrappers: the constitution precedes the requirements, convergence closes the implementation, and review is concentrated at the gates.*

## The flow: the four-phase core and its two wrappers · ESTABLISHED

The core of the method is a sequence of four phases, each with its document: requirements (what is built), design (how), tasks (in which atomic units), and implementation (the execution). Around that core two wrappers have settled. Before the requirements, the project principles are established in a constitution, and after the implementation, the result is checked for convergence with what was specified.

**Why it is here now.** The four-phase core is the stable backbone of the method and reflects the minimum logical sequence for going from an intention to code when the builder is an agent. The wrappers are the novelty that forces the card to be reformulated: GitHub Spec Kit, the most adopted implementation, today documents six steps that begin with the constitution and end by converging the implementation with the specifications, with added commands for clarification, analysis, and checklists. Kiro arrives at the same place by another path, with its requirements-first or design-first flows and an optional deep analysis of the requirements before moving on to design.

**Where to look.** GitHub Spec Kit · Kiro — spec best practices

## The project constitution · CONSOLIDATING · *new since 2026*

The constitution records what is decided once for the whole project, as opposed to what is decided in each increment. It is the founding rulebook that aligns the life cycle: project scope, domain context, technology versions, code standards, and repository structure. It persists above individual specs and is what gives continuity to judgment between different agent sessions.

**Why it is here now.** It has gone from an implicit practice to a named artifact with its own step in the method: in Spec Kit it is the first of the six and precedes the requirements. Thoughtworks teams working with SDD on existing codebases report what a constitution that really works contains, and training in the field already teaches it as the first competence, before feature specs, because it is what preserves context between sessions and reduces cognitive debt.

**Where to look.** Thoughtworks Technology Radar — GitHub Spec Kit · DeepLearning.AI — Spec-Driven Development with Coding Agents

### Approval gates · ESTABLISHED

They are the points where the person exercises judgment between phases and answers three questions: whether the requirements are correct, whether the design is viable, and whether the tasks cover the design. They concentrate review where the information is worth most and the cost of correction is lowest, instead of supervising every action of the agent. The guiding principle comes down to reviewing at the gates, not during implementation.

**Why it is here now.** It is what makes it viable to delegate the build without losing control, and it avoids the approval fatigue of validating micro-decisions one by one. The evidence of 2026 adds a counterpoint worth knowing: DORA identifies manual review as the bottleneck that saturates when the speed of code generation rises, so a badly sized gate stops protecting and starts slowing down. Morris has explored along the same lines how people and agents are distributed within the loops of software engineering.

**Where to look.** Kief Morris — Humans and Agents in Software Engineering Loops · DORA — ROI of AI-assisted Software Development

### Atomic tasks, waves, and atomic commits · CONSOLIDATING

The design is broken down into small implementation units that can be verified separately, each with its structured prompt. Independent tasks are grouped into waves that can run in parallel, and each one closes with its own commit. The agent's work thus stays reviewable piece by piece and reversible without dragging the rest along.

**Why it is here now.** It is the unit of work that makes the implementation phase supervisable without reading all the code at once, and the one that makes it possible to discard a badly solved task without undoing the whole increment. Granularity is still a matter of professional judgment more than of closed consensus, hence its status.

**Where to look.** Anthropic — Building effective agents · GitHub Spec Kit

### Specifying over an existing codebase (brownfield) · CONSOLIDATING

In a project with history, the spec does not describe the whole system: it describes the delta. Before writing a requirement it is worth answering which files will be affected, which patterns and dependencies already exist, and what must not be broken, and then fixing what exists so the agent does not overwrite it. A complete specification of the system is impracticable at scale, and unnecessary besides.

**Why it is here now.** It is where the real usefulness of the method is at stake, because most professional work happens on code that already exists. Thoughtworks teams report using SDD mostly in brownfield environments, precisely where unclear intent and hidden assumptions generate rework; Kiro offers to generate specs of what is already built, and OpenSpec declares itself outright brownfield-first. A note on terminology: the previous version of this map called this prior analysis an “Impact Report”, a name that is not used in the field or in the tools' documentation, and which this edition withdraws.

**Where to look.** OpenSpec · Kiro — spec best practices

# Block C — Writing good specs

*The core competence of the topic: producing specifications that an agent can execute without guessing. This is the part of the work that no tool delegates.*

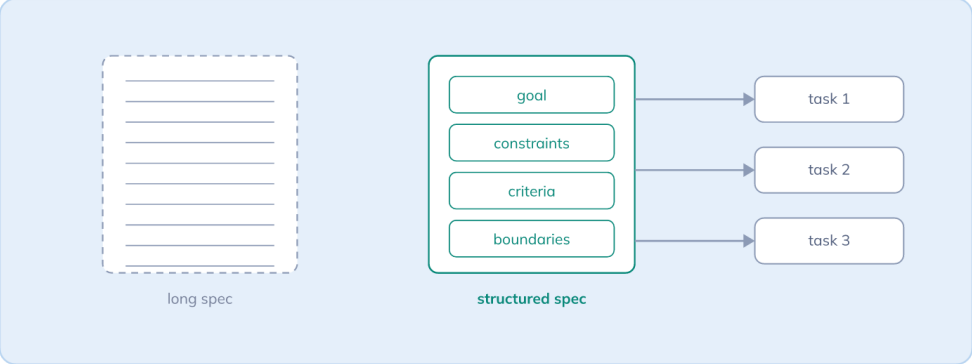



***Figure 4.*** *The smart spec, not the long one: each task receives the portion of the spec that belongs to it, not the whole document.*

## The smart spec, not the long one · ESTABLISHED

A good spec is not measured by its length. The principles that have settled are to start from the high-level vision and let the AI elaborate the detail, structure the document as professional material, feed each task the relevant portion and not the whole spec, build in self-verification and constraints, and treat it as a living document.

**Why it is here now.** It is the competence that most differentiates the result, and the one that least depends on the tool chosen. Osmani set the reference principles and has kept developing them in his work on agentic engineering. In April 2026, Zhang and Xia added a further turn with structured-prompt-driven development, which treats the prompt as a versioned and governed artifact instead of something generated once and discarded, and contributes a seven-part template for building it.

**Where to look.** Addy Osmani — Agentic engineering · Zhang and Xia — Structured-Prompt-Driven Development

## The EARS notation for acceptance criteria · CONSOLIDATING

EARS (Easy Approach to Requirements Syntax) structures natural language just enough to remove ambiguity without losing readability. Its basic template fits together precondition, event, and expected system response, and it defines five patterns that cover most requirements. A criterion in EARS is verifiable by construction.

**Why it is here now.** Agents respond well to structured requirements with consistent patterns, because the fixed form reduces the room for interpretation. The notation was born in the aerospace sector more than a decade ago and has returned to the foreground by being integrated into the SDD toolkit, notably in Kiro. Its status reflects that adoption is growing fast without yet being universal, and that it is used where it genuinely reduces ambiguity and not as an obligation.

**Where to look.** Alistair Mavin — EARS · Kiro — spec best practices

### The boundaries system: Always / Ask First / Never · CONSOLIDATING

Boundaries declare what the agent may do on its own, what it must ask about first, and what it must never do. They operate at two levels: in the project constitution they set general rules, and in each task they add specific constraints. They are the difference between delegating and neglecting.

**Why it is here now.** It is the mechanism that makes the agent's autonomy safe in the implementation phase, and the one that allows that autonomy to be raised without raising the risk. The ecosystem's support is no longer a matter for each tool: AGENTS.md has become the closest thing to a universal standard for agent instructions, some twenty tools read it natively and more than sixty thousand projects have adopted it, and its specification is stewarded by the Agentic AI Foundation of the Linux Foundation.

**Where to look.** AGENTS.md · Agentic AI Foundation

### The curse of instructions (instruction overload) · ESTABLISHED

Adding instructions to an agent does not improve its behavior indefinitely. Past a certain point, more context degrades the ability to retrieve what matters, and what sits in the middle of a long window is what is remembered worst. The countermeasure is to give each task the portion of spec it needs and to systematize reusable context in dedicated files instead of fattening a single document.

**Why it is here now.** The phenomenon has gone from practical observation to a named discipline. Context engineering is now distinguished from prompt engineering, because the question is not how to word the instruction but what should occupy the window at each moment and what should be evicted, compressed, or delegated. It also has research of its own on context degradation and on what gets lost in the middle. In team practice, reusable institutional knowledge (house conventions, deployment, in-house patterns) is encapsulated in pieces the agent loads when it needs them. That double settling, of the name and of the evidence, is what raises the card to established in this edition.

**Where to look.** Anthropic — Effective context engineering for AI agents · Thoughtworks Technology Radar — GitHub Spec Kit

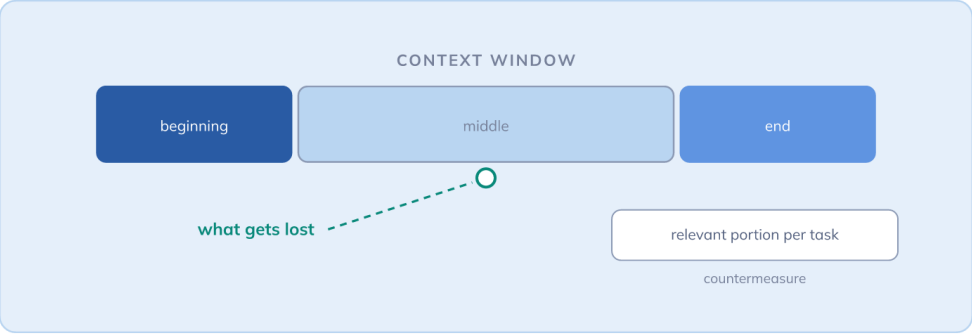



***Figure 5.*** *The curse of instructions: what sits in the middle of a long context window is what is remembered worst, and the countermeasure is to give each task its relevant portion.*

## Living specs, drift, and the Clarity Gate · CONSOLIDATING

The main risk of a spec is drift away from the code that implements it, because an outdated source of truth is worse than none. Against that, it is worth separating what is current truth from what is only a change proposal, and archiving what is already completed. The Clarity Gate is the quality test that checks it: if a different agent, or a new session of the same agent, cannot generate equivalent code by reading only the spec, the spec is incomplete.

**Why it is here now.** It is the problem that decides whether SDD adds value at six months or only in the first week. The signal this map anticipated in its previous edition has resolved in the negative: the level at which the spec is the only source and the code is fully regenerated is still experimental, with Tessl's engine in closed beta and the company itself shifting its weight toward the specification registry and governance. Meanwhile a simple, practical pattern has appeared, OpenSpec's, which keeps the directory of current truth and that of proposed changes separate.

**Where to look.** OpenSpec · Tessl

# Block D — SDD in the team and its ecosystem

*How the method fits into a real agile team, what obligations surround it, how it fails when it fails, and what it is implemented with today.*

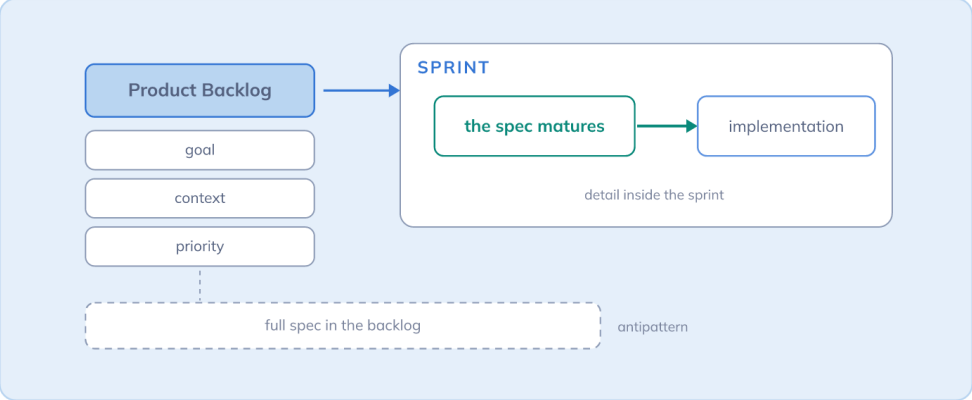



***Figure 6.*** *Where the spec lives: the Product Backlog agrees on goal, context, and priority, and the contract matures inside the sprint.*

## Fit with agile roles and ceremonies · CONSOLIDATING

The spec does not replace the backlog or refinement. In the backlog and in refinement the team agrees on goals, context, and priority; the detailed contract matures inside the sprint, as part of doing the work. The most common integration mistake consists of taking the complete spec to the Product Backlog, which turns it into a document that expires before it is executed.

**Why it is here now.** It is the practical question every team already working with Scrum or Kanban asks, and where usable doctrine has appeared during 2026. The academic work in the field also describes the tension the team has to manage: individual efficiency rises while the team's review capacity degrades and stability drops, and it proposes specifications as the governance of agent behavior at team scale, with a double harness of technical controls around the agent and methodological governance around the team.

**Where to look.** SDD for Agentic Software Engineering (arXiv, 2026) · Yuval Yeret — a step forward or back?

## Traceability and human oversight as an obligation · EMERGING · *new since 2026*

Since August 2, 2026, the high-risk framework of the EU AI Act has applied, requiring automatic logging of the system's operations, human oversight assigned to people with the necessary competence and authority, and transparency about generated content. The scope should be made precise: what the AI Act governs is the high-risk AI system the team builds, not the use of AI to program it.

**Why it is here now.** It appears in the map because it changes the value of two pieces the method already had. When the product enters the high-risk field, the chain that runs from requirement to code and the documented approval gates stop being merely good

engineering practice and become usable material for the conformity file. The relationship is one of usefulness and not of direct obligation, and its status reflects that professional practice around this intersection is yet to be built.

**Where to look.** EU AI Act — Art. 12, record-keeping · Art. 26, obligations of deployers

## Antipatterns: specification theater and zombie documentation · CONSOLIDATING

The characteristic failure modes of badly applied SDD are four. Over-specification invests more time in the spec than in the software; gates become bottlenecks; specification theater produces specs nobody really reviews; and zombie documentation leaves specs nobody consults or updates. Recognizing them is part of mastering the method.

**Why it is here now.** They are the points where SDD stops adding value and starts getting in the way, and they appear easily if there is no calibration with the principle of proportionality. External critique describes them without mercy and deserves to be read first-hand: huge documents, loss of the code's context, duplicated review work, and a false sense of security. Böckeler has asked the same question about the agent loop when examining whether test-driven development inside that loop provides real value or only theater, and the parallel is instructive.

**Where to look.** Marmelab — SDD: The Waterfall Strikes Back · Böckeler — TDD inside the agent loop

## The tool ecosystem · EMERGING

The toolkit grows fast and the specific list expires before the method does, so what is worth learning is the selection criteria. The ones that separate tools today are four: whether the flow is light or full, whether the work is on a new project or on existing code, whether a CLI inside the repository or an integrated environment is preferred, and how much governance and traceability is needed. The underlying warning stands: SDD is the method, not the tool.

**Why it is here now.** It is the most volatile area of the topic and also the one that gives the clearest sign that the method is becoming institutionalized. The reference implementation published its version 1.0 in August 2026, on turning one year old, with a revealing rationale: it declines to promise stability and claims adaptability. Around it, options with different profiles have taken shape, from integrated environments to deliberately light tools aimed at existing code, and the method itself already has an entry in the Technology Radar, formal training, and academic literature. Any snapshot of the landscape, including this one, should be checked against the comparisons that are kept up to date.

**Where to look.** GitHub Spec Kit — releases · Böckeler — Understanding Spec-Driven Development

# A note on the references

*The links included were available and verified as of the update date. Given the pace of change in this area, some may change; each revision of the map also updates its references.*