ChatGPT

Spec-Driven Software Development with Coding Agents

Overview of Spec-Driven Development (SDD)

Spec-Driven Development (SDD) is an emerging methodology where a structured **specification** (written in natural language, often Markdown) is created before writing any code. The spec serves as a shared "source of truth" for both humans and AI coding agents [1]. In practice, an AI agent (or team of agents) uses the spec to generate plans, task breakdowns, and ultimately code that implements the requirements. The goal is to bring more rigor and context to AI-assisted coding, avoiding the ad-hoc "prompt-by-prompt" *vibe coding* style that can lead to inconsistent results [2].

Several frameworks have recently emerged to facilitate SDD. They differ in how they structure the spec, how autonomous the AI agents are, and how they integrate into developers' workflows. The table below compares key SDD frameworks on architecture, agent autonomy, spec format, integration, and adoption:

Comparison of Leading SDD Frameworks

Framework	Architecture & Autonomy	Spec Format & Workflow	Integration & Use Case	Adoption/Status
BMAD Method (open-source)	Multi-agent orchestration simulating an agile team (AI "Analyst", "Architect", "Dev", etc.) for *fully* guided development [3] [4]. Human is in the loop to review artifacts.	Spec is broken into multiple Markdown docs (e.g. PRD, architecture, user stories). **Two-phase workflow**: (1) *Agentic Planning* – AI agents produce requirements & design docs; (2) *Context-Engineered Development* – AI "Scrum Master" breaks stories into tasks, "Dev" agents implement code [5] [6].	Provided as CLI (Node.js) and NPM package; can plug into any IDE or AI assistant. Suited for **large, complex projects** needing comprehensive planning and role separation [3] [7]. Typically used when building a feature from scratch or a major module with rigorous oversight.	**19k+ GitHub** ☆ (active community) [8] [9]. Mature (v4 stable; v6 in alpha) with documentation and Discord support. Used by early adopters to manage end-to-end AI-assisted feature development.

Framework	Architecture & Autonomy	Spec Format & Workflow	Integration & Use Case	Adoption/Status
GitHub Spec-Kit (open-source)	Single-agent guided workflow (LLM as coding assistant) with **human-gated stages**. Developer uses CLI commands to advance through phases [10]. Not fully autonomous – AI generates outputs at each phase for human review.	Spec is composed of multiple Markdown files generated in a **4-stage pipeline**: **Specify** (requirements), **Plan** (design/tech notes), **Tasks** (LLM decomposes plan), **Implement** (LLM writes code) [11]. Also uses a global `constitution.md` (project standards & rules) to guide the AI [12] [13]. Each phase produces checklists for validation.	Distributed as a CLI (`specify`) and VS Code extension. Integrates with GitHub Copilot, Claude, Cursor, etc., making it **agent-agnostic** [14]. Best for **well-defined features or new projects** where enterprise-grade rigor and repeatable process are needed. Developers review specs and plans at checkpoints, then let the AI implement in small, scoped tasks [15] [16].	**39k+ GitHub** ⭐ (backed by GitHub/ Microsoft) [17]. High traction in industry pilots; enterprise-friendly with Copilot X integration. Seen as a way to standardize AI coding workflows across teams.

Framework	Architecture & Autonomy	Spec Format & Workflow	Integration & Use Case	Adoption/Status
AWS Kiro (proprietary)	Integrated **IDE built on VS Code** with embedded multi-agent AI. Kiro's agents autonomously generate and maintain spec artifacts in the background. Developer remains in control via the IDE UI [18] [19]. Highly automated and tightly integrated.	Spec is managed as part of the project in Kiro: e.g. it auto-generates *Requirements.md*, *Design.md*, *Tasks.md* for a new feature [20]. **Workflow**: when a dev starts a feature, Kiro's agents draft requirements, design, and task list; the dev reviews/edits these, then agents implement and even update tests/docs via *hooks* on file changes [20]. The spec lives with the code (spec-anchored approach).	Provided as a **full IDE** (Code OSS fork) rather than a plugin [18]. Integrated with AWS ecosystem and Anthropics Claude models (Claude Code "Sonnet") for coding [21]. Suitable for teams wanting an **all-in-one AI development environment**. Common use-case: going from a high-level product idea to production code entirely within Kiro, leveraging its automation for specs and tests.	**AWS public preview** (2025). Gaining interest in enterprise circles (built by AWS) [22]. Not open-source; targeted at professional developers seeking maximum AI assistance with guardrails. Still evolving through feedback in pilot projects.

Framework	Architecture & Autonomy	Spec Format & Workflow	Integration & Use Case	Adoption/Status
Tessl Framework (private beta)	CLI-based framework + MCP (Model Context Protocol) **server** for multi-agent workflows. Aims for *spec-as-source* autonomy: in the ideal case, human edits spec, agents fully generate code. Currently requires developer oversight as it's experimental.	Spec is written in a custom Markdown with tags (like `@generate`, `@test`). Uniquely, Tessl aspires to make the spec **the primary artifact**: one spec file maps to one code file, which is generated and updated from the spec [23] [24]. **Workflow**: `tessl` CLI can even *reverse-engineer* a spec from code (`document` command) and regenerate code from spec (`build` command) [25]. It supports spec-anchored iteration: code carries a header "// GENERATED FROM SPEC – DO NOT EDIT" [24].	Runs as a CLI (and doubles as an MCP server) that works with editors like Cursor or VS Code [26]. Targeted for **future-looking teams** exploring true spec-driven automation, and for maintaining **live documentation** of code. Example scenario: maintain a critical module by editing its `.spec.md` file and letting Tessl sync changes into code, reducing manual coding.	**In beta** (by Tessl startup) [22]. Not yet open-sourced; limited trials. Tessl is distinctive for pushing the envelope toward *spec-as-source* (human never directly writes code). Industry is watching its progress, but it's not widely adopted yet due to its early stage.

Framework	Architecture & Autonomy	Spec Format & Workflow	Integration & Use Case	Adoption/Status
OpenSpec (open-source)	Lightweight CLI toolkit (single-agent assisted). Emphasizes developer control and minimal AI autonomy – the AI assists in drafting and coding per change request, but each step is guided.	Spec is just **plain Markdown** focusing on incremental change. **Workflow**: 1) *Draft Change Proposal* (create a spec describing one feature or bugfix) → 2) *Review & Align* (iteratively refine spec with the AI) → 3) *Implement Tasks* (AI generates code for the agreed spec) → 4) *Archive* (spec gets saved in a versioned `specs/` folder) [27]. Designed to bring determinism and auditability to one change at a time [28].	Distributed as a Node.js CLI. Integrates with various code assistants via simple slash-commands or an `AGENTS.md` config [29]. Ideal for **continuous maintenance or legacy ("brownfield") work** [28] – e.g. adding a feature to an existing large codebase. It avoids heavy upfront docs; instead it records each change's spec for future reference (audit trail).	**4k+ GitHub** ⭐ [30]. Frequent updates and focused on real-world developer pain points (e.g. working in large mature codebases) [30]. Early adopters in open-source and companies have praised its low overhead for day-to-day changes.

Framework	Architecture & Autonomy	Spec Format & Workflow	Integration & Use Case	Adoption/Status
PromptX / AgentOS (open-source)	Flexible **agent orchestration platform** – not a fixed SDD workflow, but a *context management system*. Allows creating custom AI "personas" and tools, treating AI more like collaborating experts than code-generators [31]. Autonomy is conversational: agents interact continuously with guidance rather than strict phases.	No formal spec document; instead context is provided via **conversational commands and memory**. Key components: **Nuwa** (define an AI persona in one line, e.g. "fintech-focused PM") and **Luban** (integrate external tools/APIs for the agent) [32]. **Workflow**: Developers engage in a dialogue with one or multiple specialized agents, feeding requirements in natural language. The platform ensures the conversation and tools maintain context (via an underlying MCP server). This is more free-form, suitable for dynamically evolving specs.	Runs as an MCP server (JavaScript/TypeScript) that injects context into AI coding tools (Claude, Cursor, etc.) [33]. Can be deployed in various ways (client lib, Node server, Docker). Best for teams that are **happy with their current coding assistants but need richer context and tool integration** – e.g. connecting an AI coding agent with a company's databases, APIs, or custom knowledge base during development.	**3k+ GitHub** ⭐ [34]. Considered *"plumbing"* for advanced AI DevOps – it's used in some innovative projects to supercharge assistants rather than replace workflows. Not as widely adopted as SpecKit/BMAD, but it's influencing how developers think about AI agent **interoperability** and customization.

Framework	Architecture & Autonomy	Spec Format & Workflow	Integration & Use Case	Adoption/Status
Claude Autonomous Coding Agent (Anthropic demo)	Two-agent pattern demonstrating *fully autonomous coding*. Uses a coordinator (initializer) agent + a coding agent. Minimal human input beyond initial instructions.	Example spec format: a simple **feature list or backlog** (in plain text or code comments) that the initializer agent reads. **Workflow**: The initializer agent analyzes the feature list, plans the project structure and tasks, then hands off to the coding agent. The coding agent writes code, creates files, and uses tools (like running tests or using Git) in iterative sessions, persisting progress to a repo [35]. It continues feature-by-feature until completion.	This is a **reference implementation** (Python script using Claude's Agent SDK). It interfaces with a Git repo and development environment to apply changes. Suited for **experimental or small-scale projects** – for example, generating a simple app from scratch given only a list of requirements, with no human intervention in coding. In practice, it illustrates what fully autonomous agents *could* do in a development workflow (e.g. auto-fixing bugs it finds) [36] [37].	**Part of Anthropic's open quickstart repo (12k☆)** [38]. This is not a product but a popular **prototype**. It's used by AI engineers to learn patterns of autonomy. Industry veterans see it as a preview of what future AI-powered development might look like, though not yet a day-to-day tool due to unpredictability.

Table: Key spec-driven coding frameworks and their differentiators (architecture, spec approach, integration, and adoption status).

Framework Highlights and Usage Scenarios

BMAD Method (Breakthrough Method for Agile AI Development)

BMAD is a comprehensive framework that **mimics an agile development team** using AI agents [3]. It comes as a CLI (NPM package) that sets up a project with roles like *Analyst*, *Product Manager*, *Architect*, *Scrum*

Master, *Developer*, and *QA* – each role performed by an AI agent with specialized prompts [4] [39]. The developer orchestrates these agents through BMAD's commands, reviewing outputs at key points.

- **Workflow:** *Phase 1: Agentic Planning* – e.g., you provide a high-level feature idea, and BMAD's Analyst agent generates a market/feature brief, the PM agent produces a **Product Requirements Doc (PRD)** with user stories and acceptance criteria, and the Architect agent drafts a design/architecture doc [40] [39]. Once approved, *Phase 2: Development* – a Scrum Master agent breaks the plan into detailed stories/tasks and a Developer agent writes the actual code for each task, followed by a QA agent validating requirements [41] [42]. The process yields a rich set of artifacts (specs, code, tests).
- **Use Case:** BMAD shines in **greenfield projects or complex features** where upfront planning pays off. For example, a team starting a new microservice could use BMAD to generate a complete spec and design before coding, ensuring clarity on requirements. It's also applied in non-software domains (documentation, creative writing) by leveraging the multi-agent brainstorming it provides [43] [44]. However, for trivial changes, BMAD can be overkill.
- **Adoption:** With ~19k stars on GitHub and an active Discord, BMAD has a strong community [8]. Early users report it helps keep large AI-generated projects organized, though it depends on having powerful underlying LLMs. It's evolving quickly (v6 in alpha) and often compared with Spec-Kit in terms of ambition [45].

GitHub Spec-Kit

Spec-Kit is a toolkit from GitHub that imposes a **structured, four-step development cycle** for AI coding assistants [11]. It's installed as a CLI which scaffolds your project with template files and scripts, and you interact with it via special slash commands in VS Code or your chat-based AI assistant.

- **Workflow: 1) Specify:** you write a high-level spec (in natural language) describing *what* to build and *why*. The AI assists by expanding it into a structured spec file (with checkboxes for clarifications, quality checks, etc.) [46] [47]. **2) Plan:** the AI produces a design/tech plan (e.g. data models, API design, architecture notes) based on the spec [48] [49]. **3) Tasks:** the AI breaks the plan into bite-sized implementation tasks (like a to-do list of code changes) [50]. **4) Implement:** finally, the assistant writes code task-by-task. After each stage, the developer reviews and approves the content (e.g. editing the spec or plan if the AI made mistakes) [16]. This gated workflow ensures human oversight at spec and design phases, preventing the "black-box" bulk code drop problem [51] [16].
- **Use Case:** Spec-Kit is used in scenarios where **predictability and consistency are paramount**, such as enterprise software teams adopting AI assistance in a controlled way. For instance, a senior engineer at a large company might use Spec-Kit to implement a new feature request: the spec step forces clarification of requirements (reducing misunderstandings), and the plan step ensures architectural fit [52] [53]. It's also useful for onboarding AI in teams – juniors or less experienced AI users can follow the structured steps to avoid jumping straight to code generation.
- **Adoption:** Spec-Kit gained enormous traction (nearly 40k stars) soon after launch [17]. Being GitHub-backed gives it credibility and it's already integrating with GitHub Copilot X. Many tech leads see it as *"Copilot with guardrails"*, turning code generation into a repeatable process. That said, some developers find the volume of markdown files and overhead too high for small tasks [54] [55]. GitHub is iterating rapidly on Spec-Kit based on community feedback.

AWS Kiro

Kiro is Amazon's entry into SDD – a **fully integrated development environment** that builds the spec-driven workflow into the tooling itself [18]. It's essentially a VS Code derivative with AI agents running behind the scenes to assist in specification, coding, and testing, all within the IDE.

- **Workflow:** When you create or select a feature in Kiro, it automatically spins up spec documents. For example, a developer starting a "user referral" feature would see Kiro generate `Requirements.md` (with user stories and criteria), `Design.md` (with proposed solution outline), and `Tasks.md` (checklist of coding tasks) [56] [20]. The developer can edit these specs as needed. Then, as code is written (either by the developer or by asking the AI to implement tasks), Kiro's agents will perform supportive actions — e.g. auto-updating tests or documentation when code changes (via *hooks*) [19]. Kiro essentially keeps spec and code in sync in real-time. The developer approves major changes but doesn't have to copy context around; the IDE manages it.
- **Use Case:** Kiro is suited for developers who want **deep AI integration without stitching together multiple tools**. A tech lead could use Kiro to manage a project from concept to completion: brainstorm requirements in plain language, let Kiro generate a baseline design, then iterate code with the AI inside the IDE. Because it's tightly integrated, it's efficient for **iterative development** – e.g. making a change to requirements will prompt Kiro to adjust the plan and suggest code changes. It's being positioned as a way to "move faster" by keeping focus on high-level intent while the AI handles boilerplate and updates.
- **Status:** Kiro is in preview (as of 2025) and backed by AWS, which suggests it may tie into AWS CodeCatalyst or cloud services. It reportedly uses Claude models under the hood for code generation [21]. Early testers (often AWS partners) have noted its impressive automation, but also the heavy Amazon flavor (optimized for AWS stacks). It's not open source, so adoption is currently limited to those in the preview program. If it matures, it could become a go-to solution for enterprises that trust AWS's tooling.

Tessl Framework

Tessl is an ambitious framework that goes beyond spec-first and aims for **spec-as-source** development. Currently in private beta, it is a CLI tool and MCP server that allows a spec to effectively *generate and round-trip code*.

- **Workflow:** You start by writing a `.spec.md` file describing a module or feature (including intended interfaces, behaviors, and tests using special tags) [57]. Running `tessl build` invokes the LLM agent to generate the code file for that spec [25]. If you edit the spec and rebuild, the code updates. Tessl can also take an existing code file and produce an initial spec for it (`tessl document`), which is useful for bringing legacy code under spec control [58]. The one-to-one mapping (one spec per code file) is a unique approach to ensure clarity and reduce ambiguity in generation [24]. Essentially, Tessl treats the spec like the "source of truth" and code as a compiled artifact.
- **Use Case:** Because it's experimental, Tessl is being tried in scenarios like **refactoring or module augmentation** – e.g. you have a complex algorithm in code and want to safely improve it: you generate its spec, refine the spec (adding clarity or new requirements), then regenerate the code. The *spec-anchored* nature means you can store these `.spec.md` files in version control and have an auditable history of *why* code changed, not just how. It's also an interesting fit for teams exploring

code generation repeatability – running the same spec through Tessl multiple times to ensure deterministic outputs, thereby highlighting where specs need to be more precise [59].

- **Status:** Tessl is not yet public, but it's notable for pushing the boundary. Thoughtworks' analysis identified Tessl as the only current tool explicitly targeting spec-as-source levels of SDD [24]. The developers are still refining the model (e.g. how to map one spec to multiple files, how to handle non-determinism). While not widely used yet, it represents a possible future where developers primarily maintain specs while AI consistently regenerates the code – a radical shift in software engineering if it can be made reliable.

OpenSpec

OpenSpec takes a pragmatic, lightweight approach to spec-driven development. It recognizes that most software work happens on **existing codebases** rather than starting from scratch, so it optimizes for incremental changes (1→n development) [28].

- **Workflow:** Suppose a developer needs to add a new API endpoint to a large application. With OpenSpec, they'd create a Markdown file under `openspec/changes/` describing the change (the spec might include the new endpoint's purpose, request/response format, any acceptance criteria, etc.). They run the OpenSpec CLI to engage the AI assistant, which reads the spec and may ask for clarifications (interactive refinement) [27]. Once the spec is agreed upon, the developer triggers the implementation step – the AI writes the necessary code changes (new functions, tests, docs) to satisfy the spec [60]. After reviewing the diff, the developer merges the code **and archives the spec** to `openspec/specs/` as permanent documentation of that feature [61].
- **Use Case:** OpenSpec is used in a **continuous development workflow** – e.g., a team managing a large SaaS product can keep an "OpenSpec" for every feature or bugfix, ensuring that each change is accompanied by a human-readable spec. This is valuable for audit and onboarding: new team members or compliance reviewers can read the spec history to understand why changes were made. It's also deliberately slim – if Spec-Kit is a heavyweight, OpenSpec is lightweight; it avoids generating excessive files or multi-phase rituals. As a concrete scenario, a senior engineer might use OpenSpec when fixing a tricky bug: they document the expected behavior in a spec, have the AI propose code changes, and once it passes tests, that spec stays as a record of the fix.
- **Adoption:** With a few thousand stars, OpenSpec has a smaller but growing user base [30]. Users appreciate that it *"meets you where you are"* – you don't need to adopt a whole new IDE or process. It works via simple Markdown and fits into Git workflows. Its focus on real-world pain points (like ensuring AI doesn't stray from requirements on legacy code) has resonated with some teams [30]. It might not generate as much buzz as multi-agent solutions, but it offers a gentler learning curve for spec-driven practices.

PromptX and Agent OS Approaches

PromptX (often described alongside "Agent OS") is less about a strict spec format and more about **enriching the AI's context and tools**. It emerged from the idea that sometimes the limitation in AI coding isn't lack of spec, but lack of context or persona control.

- **How it works:** PromptX provides a **Model Context Protocol (MCP)** server that can intermediate between the AI model and your environment [62]. Developers define *personas* – for example, an "AI Test Engineer" persona might always double-check outputs against requirements – and can plug in tools or APIs that the AI can call (for instance, a database query tool if the spec involves data) [31].

Instead of writing a formal spec document, the developer engages in a conversation with these tailored agents. For example, you might say, "Hey AI PM, design the spec for a login feature" – the persona responds with a draft spec in chat. Then you say, "AI Dev, implement this" – it writes code, possibly asking the AI PM for clarifications via the MCP orchestrator. Essentially, PromptX enables a *conversational spec-driven workflow*.

- **Use Case:** This approach is used by teams who already incorporate tools like ChatGPT/Claude in their dev process and want to **scale up the AI's capabilities** without losing flexibility. A practical scenario: a startup has an AI pair programmer (like Cursor or Copilot) but wants it to adhere to company coding standards and use internal APIs correctly. They could use a PromptX-like agent to inject a "constitution" of rules (similar to Spec-Kit's constitution, but applied dynamically across all prompts) and allow the AI to call a Slack API to ask design questions from a knowledge base. The development still happens in an editor, but the AI has more autonomy to gather context and enforce standards.
- **Adoption:** PromptX is relatively niche (about 3k stars) [34], but it represents an important category of **agentic infrastructure**. Notably, Microsoft's **AG2 (AutoGen)** is a similar concept – an "agent operating system" for LLMs that developers can use to spin up multiple agents and have them talk to each other or use tools [63] [64]. These frameworks are being explored by advanced AI engineering teams (e.g. to build custom AI code reviewers, documentation agents, etc.). While not a household name, the ideas from PromptX/AgentOS are influencing how future SDD tools might allow more fluid AI collaboration instead of rigid phases.

Claude Code's Autonomous Coding Agent (Anthropic)

Anthropic's Claude Code has shown a vision of near-future coding agents: given a sufficiently detailed spec (or list of features), an AI can *attempt to build an entire application* autonomously. Their **Autonomous Coding Agent** quickstart demonstrates this by pairing two Claude agents [35]:

- **How it works:** One agent (the *Initializer*) reads a project spec – for example, a list of user stories or a simple README describing the app to build. It then creates a plan or task list. Another agent (the *Coder*) takes each task, writes code for it, and commits it to a git repository, iterating until all features are done [35]. They can run tests, find bugs, and even update the plan if new tasks emerge [65]. All of this happens in multiple sessions, with state persisted (e.g., progress tracked via the git history and an in-memory summary of remaining tasks).
- **Usage Scenario:** This is mostly an experimental setup, but it has practical uses in prototyping. Imagine a CTO wants to see if an AI can *whip up a demo product* over the weekend – they provide a spec list ("Feature 1: user sign-up, Feature 2: upload images, …"), run the autonomous agent, and it produces a working codebase that, while it might not be perfect, saves significant initial development time. In another scenario, an AI engineer might use this to automate writing boilerplate code: for instance, generating CRUD endpoints for all entities described in a spec file, without supervising each step. It's essentially AutoGPT-like behavior tailored to coding tasks, leveraging Claude's strengths in adhering to instructions.
- **Limitations:** In real software teams, a fully autonomous agent is treated with caution. Without human oversight, the agent might produce suboptimal or insecure code. Indeed, experiments show such agents sometimes miss context or produce redundant code if the spec is ambiguous [66] [67]. Anthropics' demo is meant to inspire, not yet to replace human developers. It has garnered significant attention (the quickstart repo has ~12k stars) as it illustrates what could be possible as AI models improve [38]. Industry veterans are watching these autonomous patterns to see if they can eventually handle routine coding tasks or augment continuous integration (e.g. auto-fixing simple

bugs overnight). For now, it remains a cutting-edge glimpse into AI autonomy in software development.

Conclusion and Industry Outlook

Spec-driven development frameworks are evolving rapidly, each with a unique philosophy – from BMAD's multi-agent "AI team" approach to Spec-Kit's structured pipeline, from Kiro's AI-assisted IDE to Tessl's vision of treating specs like source code. Seasoned engineers and tech leaders are experimenting with these to **boost productivity while maintaining code quality**. There is genuine enthusiasm: for example, GitHub's studies found structured AI workflows can cut task completion time by over 50% in some cases [68]. GitHub's and AWS's heavy investment in SDD tools signals that industry players see promise here.

However, it's not a silver bullet. Early adopters have noted challenges – SDD can introduce overhead like lengthy Markdown docs to review, context misses requiring manual correction, and a feeling of "Waterfall"-like formality in fast-moving projects [69] [70]. The **key is choosing the right approach for the right scenario**. Many teams find value in spec-driven methods for sizable features or critical design work, while preferring more lightweight, iterative approaches ("natural language development") for everyday small changes [71] [72].

In summary, **Spec-Driven Development with coding agents is reshaping software engineering practices**. Frameworks like BMAD and Spec-Kit are already being piloted in real workflows to enforce discipline on AI code generation, and tools like Kiro hint at a future where IDEs have built-in AI project managers. Tech leaders should compare these approaches (see comparison table above) in terms of integration effort, autonomy, and team fit. Going forward, we can expect a hybrid of these ideas to emerge – blending human creativity and oversight with AI's speed and consistency, all driven by clear specifications as the foundation of the development process [73] [74].

Sources:

- Birgitta Böckeler (Thoughtworks) – *Understanding Spec-Driven Development: Kiro, spec-kit, and Tessl* [1] [24]
- François Zaninotto (Marmelab) – *Spec-Driven Development: The Waterfall Strikes Back* [69] [70]
- Tim Wang – *Spec-driven AI coding: Spec-kit, BMAD, AgentOS and Kiro* [18] [20]
- Remy (Redreamality Blog) – *In-Depth Comparison: BMAD vs spec-kit vs OpenSpec vs PromptX* [11] [30]
- Anthropic GitHub – *Claude-Quickstarts: Autonomous Coding Agent* [35] [38]
- GMO Research – *The BMAD Method: Spec Oriented AI-Driven Development* [4] [63]

---

[1] [12] [13] [23] [24] [25] [26] [46] [47] [54] [55] [57] [58] [59] [67] Understanding Spec-Driven-Development: Kiro, spec-kit, and Tessl
https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html

[2] [6] [8] [9] [11] [14] [17] [27] [28] [29] [30] [31] [32] [33] [34] [43] [44] [50] [60] [61] [62] [73] What Is Spec-Driven Development (SDD)? In-Depth Comparison of Open-Source Frameworks: BMAD vs spec-kit vs OpenSpec vs PromptX | Redreamality's Blog
https://redreamality.com/blog/-sddbmad-vs-spec-kit-vs-openspec-vs-promptx/

3 5 7 10 18 19 20 Spec-driven AI coding: Spec-kit, BMAD, Agent OS and Kiro | by Tim Wang | Oct, 2025 | Medium
https://medium.com/@tim_wang/spec-kit-bmad-and-agent-os-e8536f6bf8a4

4 21 39 40 41 42 63 68 The BMAD Method: A Framework for Spec Oriented AI-Driven Development - GMOインターネットグループ グループ研究開発本部
https://recruit.group.gmo/engineer/jisedai/blog/the-bmad-method-a-framework-for-spec-oriented-ai-driven-development/

15 16 48 49 51 52 53 What Is Spec-Driven Development? Tools, Process, and the Outcomes You Need To Know
https://www.epam.com/insights/ai/blogs/inside-spec-driven-development-what-githubspec-kit-makes-possible-for-ai-engineering

22 56 66 69 70 71 72 74 Spec-Driven Development: The Waterfall Strikes Back
https://marmelab.com/blog/2025/11/12/spec-driven-development-waterfall-strikes-back.html

35 38 GitHub - anthropics/claude-quickstarts: A collection of projects designed to help developers quickly get started with building deployable applications using the Claude API
https://github.com/anthropics/claude-quickstarts

36 65 Quickstart - Claude Docs
https://platform.claude.com/docs/en/agent-sdk/quickstart

37 Making Claude Code more secure and autonomous with sandboxing
https://www.anthropic.com/engineering/claude-code-sandboxing

45 The Non-Coder's Guide to Claude Code, Tested on a Real CEO
https://www.reddit.com/r/ClaudeAI/comments/1ov8osu/the_noncoders_guide_to_claude_code_tested_on_a/

64 Building the Operating System for AI Agents - Podwise
https://podwise.ai/dashboard/episodes/3471982