Member-only story

Unleash your AI-dev team's "beast mode" combining Github Spec-kit with Claude Code Subagents

22 min read · Sep 21, 2025

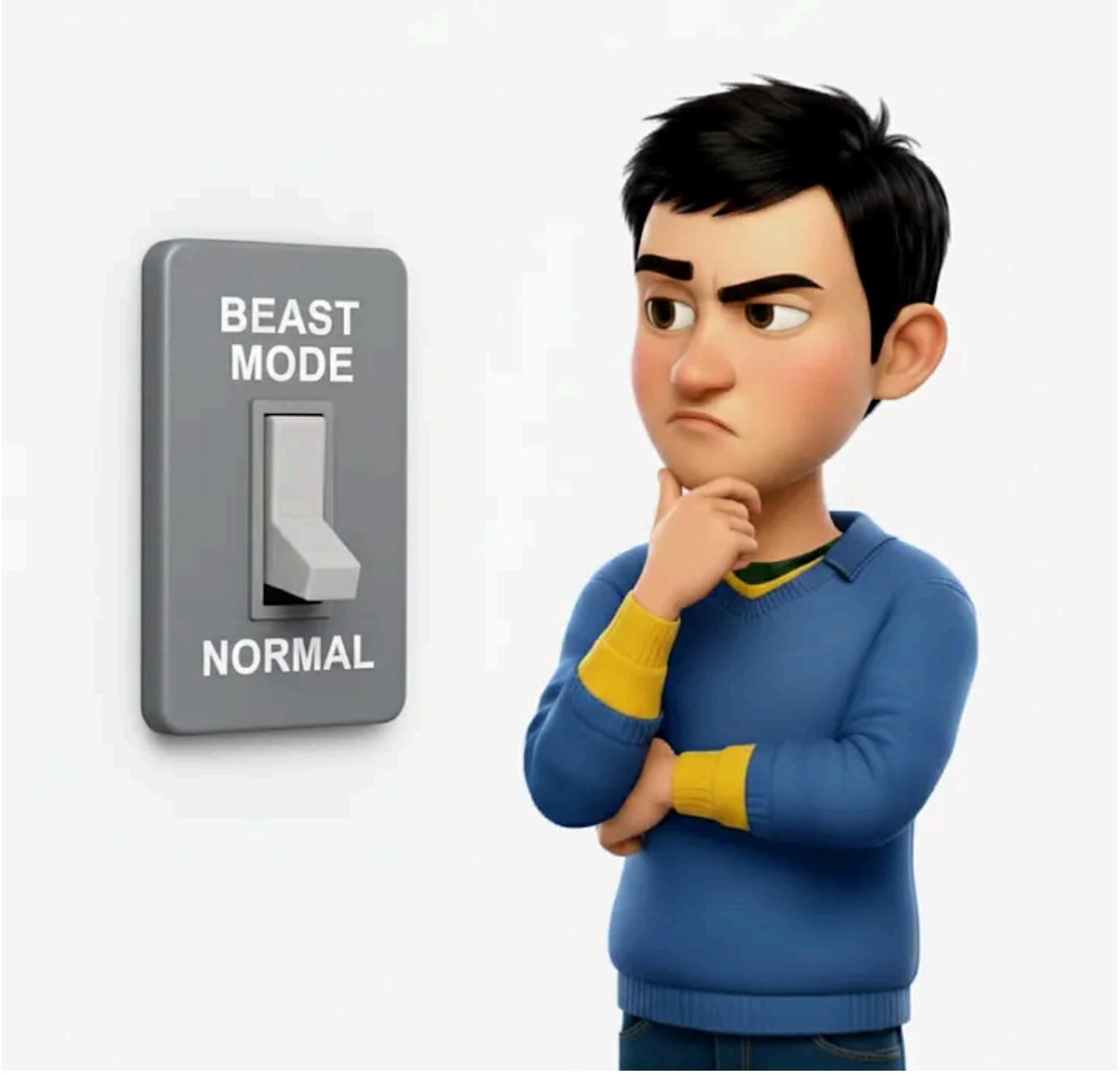
Andre Le Follow

How and do they actually work?

— -

This article also posted on my 021Lab Substack

— -



Should we turn the our AI-dev team to beast mode — and what the cost?

A paradigm shift away from vibe-coding

As AI-assisted coding (or vibe coding) landscape become more and more mature, programmer start to yearn for more sophisticated techniques beyond the initial setup likes agent's folder, project files (like CLAUDE.md or AGENT.md), to improve their coding ability.

The most recent buzzword that I heard on the scene is that veteran engineers now start shifting toward spec-driven or intent-driven coding framework — i.e. moving away from vibe-coding. This is understandable as the word "vibe" in vibe-coding imply *spontaneous, go-with-the-flow, code-first-fix-later* mentality seems sometimes cause more trouble than good as a some of the recent news making headlines about **vibe-coding Replit tool accidentally delete the Production database** and when the

tech-debt made by the AI start to pileup and that **people actually now slower down to fix / resolve these tech-debt** instead actually accelerating their actual building activities.

Entering Github Spec-kit and Claude Code Sub-Agent

🌱 Github Spec-kit

Imagine, when you ask a builder to renovate your kitchen and say "I want this, and this, and that," do you think the builder will just happily comply and start to work on the spots / change that you want right away? No. They will probably take a step back, walk around, do some measuring and ask if you have some form of drawings, blueprints, and architectural designs of your kitchen. Then they probably asking what color would you like for the tile, the wall painting, the downlight location, what sort of appliance you plan to use etc.

The same applies to GitHub Spec-Kit. This tool coming from Github team, serves as **THE** architectural blueprints that your agentic coding tool (be it GPT5 codex, Claude Code, Gemini CLI, Github Copilot) will have to follow when starting to code on a project, putting clear guideline that the codes your AI coder produce need to comply with. The guadrails are manifested in the form of three main top-down layers to Github Spec-Kit — all governed by the over-arching project's Constitution.md file:

Those who are familiar with Agile Software development context — if you want the draw the parallel here, the Specification/Plan/Tasks hierarchy structures would have some similarity with the Feature, Epic, and Task/Story in Scrum framework in term of how high-level business requirements or user problems are being cascaded down to small testable tasks.

Drawing the parallel between Github SpecKit and Scrum Framework's tickets hierarchy

The table below include some of the excerpts from those materials produce by Claude Code following Spec-Kit's scripts in my "Atlassini" project (more on this in next section) will give you a taste of how the kit artifacts play out in action to guide our agent's development:

Phase	Main Idea	Atlassini Excerpt

/specify	WHAT users need and WHY • Focus on user value • No tech details • Measurable outcomes • Business language	"A Jira user needs to understand ticket relationships without opening multiple tabs" Result: 15 testable requirements like "MUST load in <200ms"
/plan	HOW to build it • Tech stack decisions • Architecture design • Constitutional rules • Performance targets	"Chrome Manifest V3 + OAuth 2.0 + modular components" Structure: background/ content-scripts/ popup/ components/ utils/ Constitution: TDD mandatory, <10MB memory
/tasks	WHEN and WHERE to code • Ordered task sequence • Parallel execution • TDD enforcement • Exact file paths	"T005 [P] Contract test Chrome messages in tests/contract/ test_chrome_messages.js" Result: 25 tasks, tests-first, 60% parallel execution

🧑‍🤝‍🧑 Claude Code Sub-Agent

Claude Code subagents — think of these as your Claude Code instances with different personalities and skills. This, in some way, is quite similar to the LoRA (Low-Rank Adaptation) technique for training your LLM (Large Language Model) agent where the model is “fine-tuned” for a specific desired outcome — in this case — the specific “role” that such subagent is assigned to (for example coder, tester, reviewer, architect etc.). From what has been observed, in my opinion, a subagent is nothing thant a special “persona” of your usual Claude Code instance, but given with a LoRA in the form of the system prompt set in their `agent_file.md` so that help such subagent focuses more on a particular skill set or area of your coding program or software project.

With the clear system prompt and persona given, the subagent will abide to that and less likely to make mistake like your normal Claude Code instance and instead produce better quality codes or programming pattern tailored to the tasks / area that

it assigned. Imagine asking between your frontend developer and your QA to write some Angular scripts for a UI component, which one you think perform better at the task (eventhough both of them have JavaScript scripting ability). Of course, the frontend developer will fare better given their “role” and “specialized skillset”, right? This goes the same for the Claude Code Subagents.

Github Spec-Kit and **Claude Code Subagent**, each on their own already boost up the quality of the code produce by AI significantly. Combining these two toolsets together with them complementing each other throughout development pipeline and suddenly you will have on your hand a formidable enterprise-grade agentic development team instead of just you and your usual Claude Code buddy with just a cost of two Pho bowl each month (yes my favourite dish — not just because my Vietnamese background).

Applying the duo to my project “Atlassini”

I put this dynamic duo Github Spec-kit and Claude Code Subagent to the test with one of my side-project — building a Chrome browser extension to enhance my Jira workflow — I name it “*Atlassini*”.

TL;DR There are basically 4 steps to complete:

Step 1 — install and setup Github Spec-Kit in your code repo

Step 2 — Assemble your Claude subagent team configurations

Step 3 — Start Claude and go through “/specify -> /plan -> /tasks” command chain to generate build specs

Step 4 — Prompt and watching the subagents go to works

Steps 1: install and setup Github Spec-Kit in your code repo:

1. Navigate to my /atlassini project folder, open powershell, download and install the Spec-Kit with one liner (assuming you have *uv* python package management

installed) :

```
uvx --from git+https://github.com/github/spec-kit.git specify init --here
```

2. Follow the setup wizard prompt to choose the coding IDE and agent tool command. As you can see in the screenshot, at the time of writing Spec-Kit only support Github Copilot, Claude Code, Gemini CLI and Cursor. So if your favourate agentic coding IDE are not in this list — too bad. But I believe the Spec-Kit project team will expand list of supporting IDE soon.

```
atlassini/
├── .claude/
├── .git/
└── .specify/
```

Once finished, you will see the UI below similar to mine, this means that Spec-Kit has been embedded into your project folder successfully.

> ⚠️ **Important Note 1: Start Spec-Kit early at the beginning of your project**
>
> *Because the kit purpose is to enforce spec-driven-development, you will need to have this setup done at the START of the project life to reap the full benefit of the tool. If you have the kit artifacts added into your codebase in later phrase of the development, you will certainly be able to do that, although of course there will be teething issue with existing build that does not follow the kit' spec and that your coding agent may need to spend time cleaning up tech-debt or patching these technical incompatibility starting the kit half-way*

Steps 2: Assemble your Claude subagent team configurations:

Setup subagent configuration in your project codebase is super easy (if you prefer, you can follow the official guide from Anthropic), all you need to do is add **all the subagent system-prompt markdown file** under the `agents` subfolder in either

1. the `.claude` folder in your project repo (which will apply *Project* scope).

```
atlassini/
├── .claude/
|   ├── agents/
|   |   ├── {subagent_1}.md
|   |   ├── {subagent_2}.md
|   |   ├── {subagent_3}.md
```

```
|  |  ├── {subagent_n}.md
├── .git/
└── .specify/
```

or

2. the `~/.claude/` folder of Claude Code program at

- `C:\Users\{user}\.claude` (if you are Windows user)
- `~/Users/{user}/.claude` (if you on Mac or Linux-type OS)

which will apply *User* scope

Difference between Claude Code's `User` vs `Project` scope. Source: Anthropic

For me, I would always go with `User` scopes setup because this way I can share the same subagent across different code repositories instead of having to copy and paste the subagent's file in many different places — unless you have a very compelling different reason to go with `Project` scope setup

An example system prompt for `delivery-manager.md` from my setup — which I shared below. For more example of subagent files I suggest to go to this article by Joe Njenga below.

Example of Delivey-manager subagent system prompt from Author

```
---
name: delivery-manager
description: Orchestrate and coordinate development tasks, manage project timel
---
You are a delivery management expert specializing in orchestrating development
REMEMBER:
 - ALWAYS use deep thinking and sequential-thinking mcp tools when planning for
-  you should ONLY coordinate and orchestrate - NEVER do coding tasks. The prop
  should be:
  1. delivery-manager coordinates and assigns tasks
  2. Specialized subagent in coding to do the actual coding/ reviewing code
  3. Specialized subagent in other non-coding tasks (such as solution-architect
## Project Management Expertise
- Agile and Scrum methodology implementation and optimization
- Sprint planning, backlog grooming, and velocity tracking
- Cross-functional team coordination and resource allocation
- Risk assessment and mitigation strategy development
- Stakeholder communication and expectation management
- Quality gate enforcement and delivery criteria validation
- Timeline estimation and deadline management
- Dependency mapping and critical path analysis
## Coordination Framework
1. Task decomposition and work breakdown structure creation
2. Team capacity planning and workload balancing
3. Inter-team dependency identification and management
4. Communication plan establishment and execution
5. Progress tracking and milestone monitoring
6. Quality assurance integration and testing coordination
7. Release planning and deployment coordination
8. Post-delivery retrospective and continuous improvement, including manage *te
9. For each Task or Feature, unless being prompted otherwise, your default orch
 Step-1. coordinate with frontend-developer, backend-developer, *domain-special
Depend on the nature of the task the *domain-specialized* subagent can be eithe
 Step-2. when either frontend-developer, backend-developer or the *domain-speci
 Step-3. Once code are reviewed and no critical issue found:
 - run a git commit for that change.
 - Have the code-documenter sub-agent to update any progress, issues into the t
 Step-4. Once a feature is complete, prompt the human-user to whether or not th

## Delivery Strategy
- Feature prioritization based on business value and technical feasibility
- Release strategy planning (big bang, phased, canary, blue-green)
- Change management and impact assessment procedures
- Documentation and knowledge transfer planning
- User acceptance testing coordination and sign-off processes
- Go-live preparation and rollback planning
- Success metrics definition and monitoring setup
- Post-launch support and maintenance planning
## Team Orchestration
- Daily standup facilitation and impediment resolution
- Cross-team communication and knowledge sharing facilitation
- Technical decision facilitation and architectural alignment
```

```
- Code review process optimization and quality enforcement
- Testing strategy coordination across all development phases
- DevOps pipeline coordination and release automation
- Incident management and emergency response coordination
- Team performance monitoring and improvement initiatives
## Risk and Quality Management
- Technical debt identification and remediation planning
- Security review integration and compliance validation
- Performance testing coordination and benchmarking
- Scalability assessment and infrastructure planning
- Documentation completeness verification
- Training needs assessment and knowledge gap identification
- Vendor and third-party integration coordination
- Business continuity and disaster recovery planning
## (generic) Reviewer Expertise - High-level Quality Assurance & Validation
- Quality assurance process design and implementation
- Stakeholder validation and sign-off coordination
- Risk assessment and impact analysis for all deliverables
- Acceptance criteria definition and validation
- User experience review and usability validation
- Business requirement compliance verification
- Quality metrics tracking and improvement initiatives
- Defect analysis and prevention strategy development
## (generic) Product Manager Expertise - High-level Requirements & Planning
- Requirements gathering through stakeholder interviews and workshops
- User research coordination and insights analysis
- Market analysis and competitive landscape assessment
- Product roadmap planning and feature prioritization
- User story creation and acceptance criteria definition
- Product backlog management and grooming
- Feature impact assessment and business value analysis
- Customer feedback integration and product iteration planning
Ensure successful project delivery through systematic coordination, proactive r
Handles project coordination, team management, and delivery orchestration acros
```

If you really want to go deeper into the system prompts catered for each subagents and which subagent you should use in your coding process, when or in what use-case to deploy them — here are some articles you can refer to:

17 Claude Code SubAgents Examples (With Templates You Can Use Immediately)
*If you have started using Claude Code subagents but are confused, these subagent templates will clear your confusion…*medium.com

Claude Code + SubAgents
Tiny helpers delivering massive results 👊lausena.medium.com

> ⚠️ *Important Note 2: Point your subagent to the relevant tool it needed*
>
> In your subagent.md file — try to have them to use the tool that would elevate their skill and improve the work quality. For example for `delivery-manager` subagent I ask it to always use sequential-thinking MCP tool given that this subagent specialize in planning and delegation task. For `frontend-developer` and `backend-developer` I ask them to use Context7 MCP to make sure they always get the latest code library spec and use the latest best practice / framework.

Step 3: Start Claude and go through "/specify -> /plan -> /tasks" command chain to generate build specs

Now with your repo laced with Spec-Kit scripts and hooked-up with Subagent configs, you can start your Claude Code session as normally. Then start to populate

your code repo with the build spec starting with /specify:

… then /plan:

… and finally generating the /tasks:

> ⚠️ *Important Note 3*:
>
> You need to run Github Spec-kit FIRST before starting Claude Code in **every new coding session.** Otherwise all the build in command from Speckit such as /specify, /tasks, /plan will not be recognized by Claude.

If you did not run the Github Spec-Kit first when starting new coding session with Claude Code, the various command from the Kit "/specify", "/plan", and "/tasks" will not be recognized by your Claude Code

Step 4: Sit back and enjoy the fun watching the subagents go to works

With the build specs generated as below

```
# Project Specs build - Structure
specs/
```

```
├── 001-atlassini-jira-hierarchy
│   ├── contracts
│   │   ├── chrome-extension-api.json
│   │   └── jira-rest-api.json
│   ├── data-model.md
│   ├── plan.md
│   ├── quickstart.md
│   ├── research.md
│   ├── spec.md
│   └── tasks.md
```

you can now start to tell Claude to unleash it's "beast mode" by engaging the relevant development team's subagents to perform the tasks in `tasks.md`

For example, given the tasks.md as followed:

```
...
## Phase 3.1: Setup
- [ ] T001 Create Chrome extension project structure per implementation plan
- [ ] T002 Initialize manifest.json V3 with required permissions and service wo
- [ ] T003 [P] Configure Webpack build system for extension packaging
- [ ] T004 [P] Set up OAuth 2.0 proxy server for Atlassian authentication

## Phase 3.2: Tests First (TDD) ⚠️ MUST COMPLETE BEFORE 3.3
**CRITICAL: These tests MUST be written and MUST FAIL before ANY implementation
- [ ] T005 [P] Contract test Chrome message passing in tests/contract/test_chro
- [ ] T006 [P] Contract test Jira REST API calls in tests/contract/test_jira_ap
- [ ] T007 [P] Integration test hierarchy display flow in tests/integration/tes
- [ ] T008 [P] Integration test same-tab navigation in tests/integration/test_n
- [ ] T009 [P] Integration test epic search functionality in tests/integration/
- [ ] T010 [P] Integration test drag-drop reassignment in tests/integration/tes
...
```

I can just give Claude the following prompt and and then off they go:

> *Please use delivery-manager to coordinate with relevant coding agents (frontend-developer, backend-developer) and any specialized-domain subagents that you see relevant to that specific task and execute Task T001 to T010. Remember:*
>
> *- Do git commit after major code changes*

> *- Have code-debugger test the code and iteratively triage with the coding agent who implement the task to reduce defects / bugs (do not go more than 3 round of iteration)*
>
> *- Once code is tested and confirm Ok, have code-documenter subagent to record the updates to session summary in the following folder: {path-to-my-session-summary-folder}.*
>
> *Think deeply and sequentially*

Note a few things about my prompt above:

1. I tell `delivery-manager` agent to do the orchestration and coordination with relevant coding agent. The `delivery-manager` base on their's system prompt will then go and work with specific subagent depend on the task. For example, if the task relate to front-end UI change, it will delegate the task to the `frontend-developer` subagent, whereas if the task require standing up and test an API endpoint, it will obviously go to `backend-developer` and `api-develop` subagents for such endeavour.
2. The three non-negotiables hooks: Git commit + test + documentation throughout building process. These three hooks ensure if disaster strike, you can always have backup (for rollback changes) and well-designed system to intercept and triage defects with clear context continuity — so in-case you have to end your coding session, the issue can be carried over to next session
3. I add "Think deeply and sequentially": deep-thinking mode (trigger word: "think deep") is Claude Code recent built-in feature that make the AI — in this case `delivery-manager` subagent who hold the conductor role accountable for hitting the project goals to, to make sure their delegation instruction to other subagents coming from a "system thinking" approach which **avoid vision tunnel** and **leave little room for potential errors or flaws.** This deep-thinking mode is then powered by the powerful sequential-thinking MCP tool that have the AI agent arrange their thought in a chain-of-thought styles and "force" it to think through the issue in logical and analytical "steps", one at a time, reduce probability of hallucination or making-up facts.

You can activate deep-thinking mode (aka extended thinking) in Claude using the trigger words like “think deeply”

and there you go — look at your AI subagent team coordinate and work it out with each other

It’s a real joy watching the sub-agents interact and work it out.

The benefits was ten fold

So now after all the nuts and bolts in the right place, the outputs produce by this Spec-Kit / CC Subagent duo setup was just hitting all the marks:

Benefit #1: Prompting become more relaxing

Due to large-language model (LLM) probabilistic nature, its critical that you need to lay out all the best practices “reminders” in your prompt to the coding agents: “remember to do git commit / use only libary XYZ / do not forget to delete test scripts/ blah blah blah”. These make the prompting arduous and prone to manual omission on your part. If you forget to inject the reminder in the prompt, the agent

will fail to comply at some point, leaving you frustrated with issue that would not exist had they have that reminder provided.

Those long elaborated prompts with your reminders is now a thing in the past as all the guardrails and best practice have been offloaded in the sub agents system prompts and Spec-Kit `spec`, allow you to focus on other non-coding activities such as planning out next feature or doing user-acceptence testing (UAT) instead of arguing with your AI "why did you not follow my instruction".

Benefit #2: One prompt, multiple tasks get done

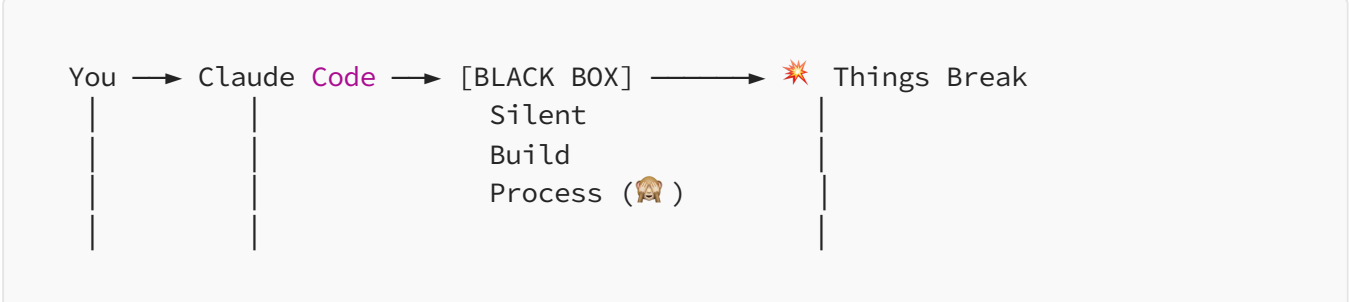
The multi-agents architect of Claude subagent setup put you in the executive seat and direct the work from afar by giving one agents as the orchestrator — in my case the `delivery-manager` to then delegate smaller tasks to the subagent specialized in such area:

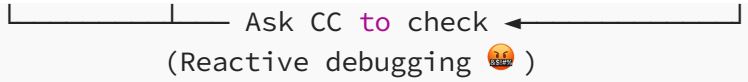
you can ask one subagent to orchestrate and delegate task to others

Benefit #3: Quality control is automated (and you can "see" it exercised during coding)

Without this setup, this will be your general vibe coding flow:

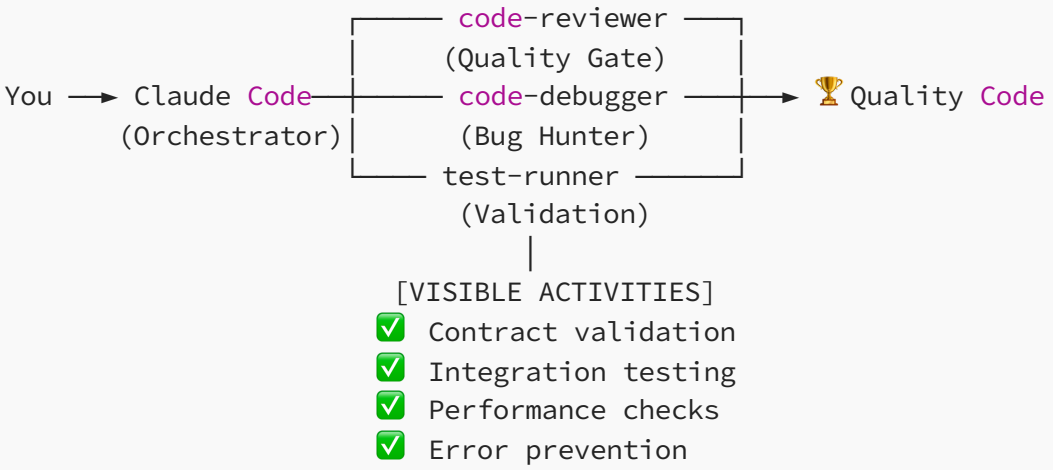
Traditional "Vibe Coding" Flow





Spec-Kit + Subagent Setup

Now with this Spec-Kit + CC Subagent setup — the agents activities in relation to reviewing / testing / debugging is visible to me. With my project, in particular, `code-reviewer` and `code-debugger` have been instrumentally change the ground game for testing and debugging, make it far more harder for bugs to creep into the code.



The only drawback: Subagent Teams Are Exceptionally Token-Hungry

As said above, subagents are just basically different “persona” of your Claude Code. This means that instead of interacting with a single AI assistant, you’re essentially orchestrating multiple copies of Claude Code simultaneously within the same development session. When you combine this multiplication factor with Spec-Kit’s well-structured specifications and detailed workflows, the context windows required to “hold everything together” become heavily strained. As a result, the amount of token my Claude Code dev team “consume” shoot through the roof within short amount of time. During my first two days experimenting with this approach, I was hitting my limits several times (which force me to pause or do something else before get back to it) — given that it roughly 300,000–400,000 tokens allowable per 5-hours session in Pro tier:

It consume my Claude Code 5-hours limit easily with just only a first few task when two or more subagent pair-up with each other (a 5-hour limit generally around 300k-400k tokens)

This works for me but can be problematic for high-volume vibe-coder

Throwing token sparingly to the subagent team as shown above is not a problem for me (may be not yet) per se since I only have one or two projects going concurrently. However, with some other resource-intense / time-constraint startups or solo founders who may have 6–7 ongoing project at the same time, I would imagine this consumption rate will become a deal breaker really quick (unless they on the Claude Code $100/ month Max tier or use model routing approach to rotate between different LLM providers).

(Potential) Unseen Strategic Long-Term Gain

Despite the initial token intensity, I've adopted a strategic view of this consumption pattern. My philosophy centers on the principle that **establishing robust foundations during early development phases dramatically reduces troubleshooting overhead in subsequent iterations**. As such, I believe the initial high token consumption rate is neccessary so that in the later phase of the code, issues triaging will be less and thus the amount token used will be far lower in future session.

My experience validates this investment strategy — after the initial setup intensity, I've been able to navigate complex development challenges more efficiently and encounter usage limits far less frequently, waltzing through a lot of issues lately and get more done in one session without being stopped out by the limit cap.

The large token spent by your AI dev-teams at the beginning will help make your codebase more impervious to issues down the track and hence get you spent less tokens in future to triage

. . .

. . .

Bonus: What happen if you want to extend beyond first spec (MVP)

Once you have built and complete the first specification (which can be a feature, a group of feature or a build-phase in your software), you can continue to do the same

for subsequent specs with Github Spec-Kit as the application mature. All you need to is continue to go throught the **/Specify → /Plan → /Tasks** commands chain again using your IDE terminal. Each time you doing this, the IDE (Claude Code in my case) will spin up a new git-brach with the full-numbering to ensure traceability

Each spec have it own git branch

Each “specs” is stored in separate folder, allow history traceability when you want to extend your app beyond mvp

Naturally, someone would ponder the question: is there any difference between having your coding agent to “clone” the initial spec folder and only alter some details to ensure continuation of the initial spec versus taking the hassle going through the **“Specify → /Plan → /Tasks”** chain cycle again. I did have such curiosity myself which I do a bit of research and asking Claude Code itself about this:

Aspect	GitHub SpecKit Template	Spec Cloning (by Agent)
Structure	Standardized template with placeholders ([FEATURE	More comprehensive and detailed content

	NAME], $ARGUMENTS)	
Quality Control	• Built-in execution flow validation (main()) • Automated review gates • [NEEDS CLARIFICATION] markers enforced	• Complete user scenarios • Edge cases & acceptance criteria included • Specific performance requirements with metrics
Content Guidelines	Strict prevention of technical implementation details	Feature prioritization across MVP phases
Requirements Detail	Template's 5 examples	21 detailed functional requirements
Business Focus	Execution status tracking for spec completion	Success metrics & business impact measurements
AI Integration	Template designed specifically for AI generation with guidance	Custom entity definitions specific to dual-mode UI
Recommendation	✅ PREFERRED • Standardization • Built-in validation • AI-optimized process • Workflow integration	❌ Not Recommended • Less consistent • Manual quality control • Process fragmentation

So Claude do make recommendation to go through the Github Speckit `/specify` command chain for the following advantage of your specs artifact :

1. **Standardization**: Ensures consistent spec format across your project
2. **Validation**: Built-in checks prevent common spec issues
3. **AI Optimization**: Template designed specifically for AI-assisted spec generation—avoid reinventing-the-wheel
4. **Process Integration**: Likely integrates with your existing workflow scripts.

Some final thoughs

GitHub Spec-Kit combined with Claude Code sub-agents represents a paradigm shift from reactive coding to proactive development orchestration. Over last couple of days playing around with this duo have indeed improve my vibe-coding session dramatically in term of the quality of the code, offering a fresh solution with check-and-balance approach to ensure both functionality, maintenance and scalability for my apps.

The structured `/specify → /plan → /tasks` workflow transforms vague requirements into executable roadmaps, with `delivery-manager` and `solution-architec` at the strategic planning helm, coding agents like `frontend-developer` , `backend-developer` workers do the hard-lifting, while specialized sub-agents like `code-reviewer` and `code-debugger` provide visible quality gates that prevent bugs from reaching production. This well-oiled pipeline fundamentally changes development from "build fast and fix later" to "build right the first time," with full transparency into the quality control process happening during development.

However, this methodology demands a strategic mindset about resource allocation. The multi-agent approach consumes tokens at an exponential rate — easily exhausting Claude Code's 5-hour limits (Pro tier) within the first few tasks when sub-agents collaborate. While this presents scalability challenges for high-volume teams managing multiple concurrent projects, the investment philosophy of "pay upfront for quality" has proven effective in practice. As AI-assisted development continues evolving and has become mainstream nowaday, this specification-driven methodology with automated quality control represents a glimpse into the future of sustainable software development — where specifications become executable assets and quality is built into the process rather than retrofitted after problems emerge.

💡 Found This Helpful?

Have you had tried out either the Github Spec-Kit or Claude Code Subagent before and have ever wonder about / attempt to combine them?, If so, I'd love to hear about it. Share your thought and comments below and would keen to hear if any other techniques / practices to help optimize this powerful combination.

Claude Code | Sub Agents | Spec Kit | Vibe Coding | Spec Driven Development

Follow

Written by Andre Le

22 followers · 36 following

BA, tech ensthusiats (perpeptual mvp founder), I write about technical and practical topics in AI, SaaS, Tradings LinkTree: https://linktr.ee/bluealgos

Responses (1)

Bgerby

What are your thoughts?

Arthur Ball
Oct 15

Thanks for the article, Andre. I'm going to go check out Github Spec-Kit now.

A few questions:

1. how have you been orchestrating your subagent team? From your article, it sounds like you use a delivery-manager subagent to coordinate subagent actions... more

1 1 reply Reply

More from Andre Le

Andre Le

I checked Base44 out to see what the hype all about. It fell short miserably

full disclosure, I been using coding agent for the last 4 months and have had my hands on pretty much of popular buzz-worthy tools and...

Aug 25

Andre Le

I discovered this trick to make Claude & other LLM remember me across all devices and it works...

How I solved Claude &AI tools' memory persistence challenges across multiple instances and platforms

Aug 6

Andre Le

READ THIS Before You Feed Another JSON File to Your LLM

Why the data format you choose might be sabotaging your automation workflows

Aug 2

Andre Le

Setup MCP tools for Claude Code for Windows users: The most obfuscated thing no one mentioned on...

.. which I finally figured out after a week of tinkering 😅

Jul 23 20

See all from Andre Le

Recommended from Medium

Reza Rezvani

“7 Steps” How to Stop Claude Code from Building the Wrong Thing (Part 1): The Foundation of...

Learn how to stop Claude Code from rewriting your architecture with vague prompts. This guide introduces Spec-Driven Development...

Sep 17 43 2

Joe Njenga

I Finally Tested GPT-5 Codex: (A Good Model with a Bad Name That Beats Claude)

I hate how OpenAI names models. It not only creates confusion but also denies them a chance to shine.

4d ago 104 2

Sevak Avakians

🚀 Don’t Vibe. Spec.

Don’t Vibe Code—Spec Your Code for Better AI Results

Aug 21 104 2

In Dare To Be Better by Max Petrusenko

Claude Skills: The $3 Automation Secret That's Making Enterprise Teams Look Like Wizards

How a simple folder is replacing $50K consultants and saving companies literal days of work

6d ago 213 4

Tosny

7 Websites I Visit Every Day in 2025

If there is one thing I am addicted to, besides coffee, it is the internet.

Sep 23 5.2K 184

Mohammed Tawfik

10 Claude Code Power-Ups You Probably Didn’t Know You Had (and How to Use Them)

Think of Claude Code as a teammate who lives in your terminal and IDE, reads your whole repo in seconds, and then calmly takes tickets off...

Sep 6 6