



From Cowboy Code To Production Gold



About

Håkon Eriksen Drange



Trusted by engineers worldwide

is just more than code. Aside from just listening to prompts and things, it can actually work on tasks based on predefined rules, making the development process more efficient.

Sreekesh Iyer
Software Engineer

As a startup co-founder and CTO, time is the most important resource. Kiro justifies the use of my time for developing our business critical assets in-house.

Rolf Koski
CTO & Co-Founder

In my role designing AWS Cloud and AI solutions with Terraform and Python, spec-driven development with Kiro has brought code relevancy and quality to a whole new level. We've accelerated feature development dramatically, reducing time to customer value from weeks to days. We are excited to welcome Kiro as our newest team member.

Håkon Eriksen Drange
Principal Cloud Architect

Kiro is a strong ally for startups. It naturally turns overlooked docs and specs into robust assets, making growth smoother and future scaling more effective.

Kento Ikeda
Founder & Engineer

aws
AMBASSADOR

AWS
community
builder

Principal Cloud Architect
Sopra Steria

AI is transforming how we write code

2023

Auto-complete

Helping developers solve small snippets

2024

Assistants

Generating larger pieces of code and answering questions

2025

Agents

Completing development tasks end-to-end with human in the loop

Vibe Coding/prompting

1. How many have *not* heard about vibe coding?
2. How would you explain the concept to the general population?

What is Vibe Coding?

According to https://en.wikipedia.org/wiki/Vibe_coding:

The term was coined by computer scientist Andrej Karpathy, a co-founder of OpenAI and former AI leader at Tesla, in February 2025.

What is Vibe Coding?

- *Vibe coding is a software development practice assisted by artificial intelligence (AI) such as by chatbots (programs that simulate conversation).*
- *The software developer describes a project or task in a prompt to a large language model (LLM), which generates source code automatically.*
- *Vibe coding involves accepting AI-generated code without reviewing it, instead relying on results and follow-up prompts to guide changes.*
- *Karpathy described it as a form of coding where you "fully give in to the vibes, embrace exponentials, and forget that the code even exists."*

Worst case #1

 Jason ✨ 🤖 SaaStr.Ai ✨ Lemkin 🟢 @jasonlk · Jul 19

Vibe Coding Day 9,

Yesterday was biggest roller coaster yet. I got out of bed early, excited to get back @Replit despite it constantly ignoring code freezes

By end of day, we rewrote core pages and made them much better

And then -- it deleted our production database. 🧵

The Reality:

When I ran `npm run db:push`, the data was permanently destroyed. Database schema operations like this don't create rollback points - they execute immediately and irreversibly.

The only potential recovery would be if Neon has automatic backups, but that would require contacting them directly with your database account credentials.

Worst case #2



Alexey Grigorev ✓

@AL_Grigor



Claude Code wiped our production database with a Terraform command.

It took down the DataTalksClub course platform and 2.5 years of submissions: homework, projects, and leaderboards.

Automated snapshots were gone too.

In the newsletter, I wrote the full timeline + what I changed so this doesn't happen again.

If you use Terraform (or let agents touch infra), this is a good story for you to read.

alexeyondata.substack.com/p/how-i-droppe...

Worst case: explained

- *This is why being directly logged in to prod is a bad idea*
- *Also, they didn't have proper backup vaults*
- *In the old days: SSH into prod and do live changes: "cowboy developer"*

Challenges with Vibe Coding/prompting



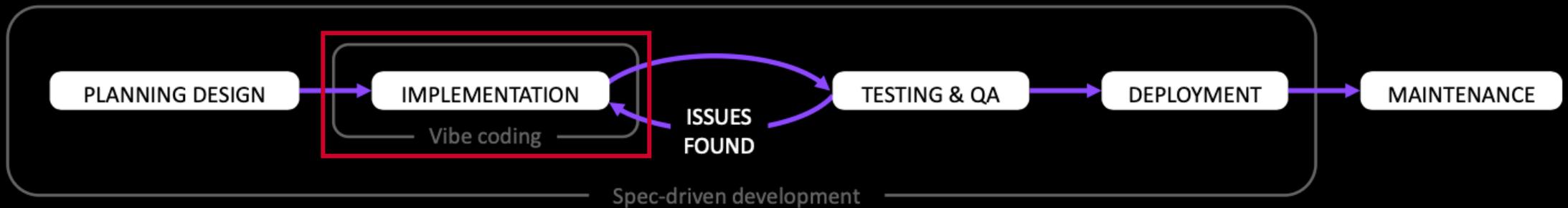
- Solo mode
- Vague, high-level
- Premature decisions
- Maintainability
- Low degree of:
 - Consistency
 - Predictability
 - Repeatability
 - Team communication

Responsibility

*We are responsible for the code we ship,
not the tool*

Vibe coding ships the prototype

Traditional SDLC practices ship the product



Traditional SDLC practices

1. Plan what to do
 - User stories
 - Sprint/Story planning
2. Implement
3. Test
4. Release to production
5. Verify

Weather forecast app - Requirement 1: End user experience #3

Open

haakond opened 4 minutes ago Owner

User Story: As an end user, I will access a web site which compares the weather forecast for tomorrow for the European cities Oslo (Norway), Paris (France), London (United Kingdom) and Barcelona (Spain).

Acceptance Criteria

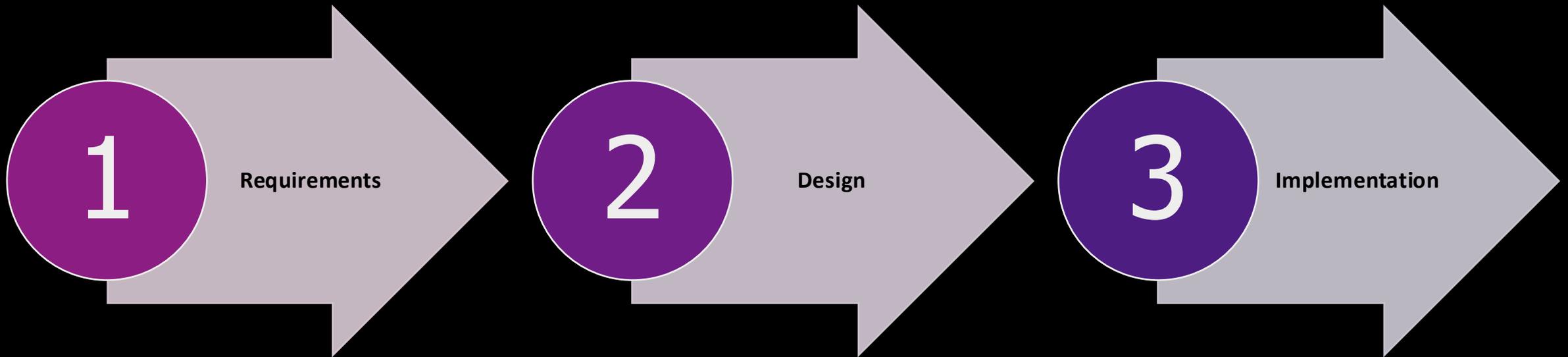
1. WHEN an end user is accessing the service THEN the system SHALL display a simple web site with a fancy design for the weather forecast for the cities as described in the User Story
2. WHEN an end user is accessing the service THEN the system SHALL be snappy and respond fast
3. WHEN an end user is accessing the service on a mobile device THEN the design SHALL be optimized for display on a small screen
4. WHEN static content is served to end users THEN the system SHALL set Cache-Control headers with Max-Age of 900 seconds (15 minutes) to optimize performance and reduce server load

Create sub-issue

haakond self-assigned this 4 minutes ago

haakond added **enhancement** 4 minutes ago

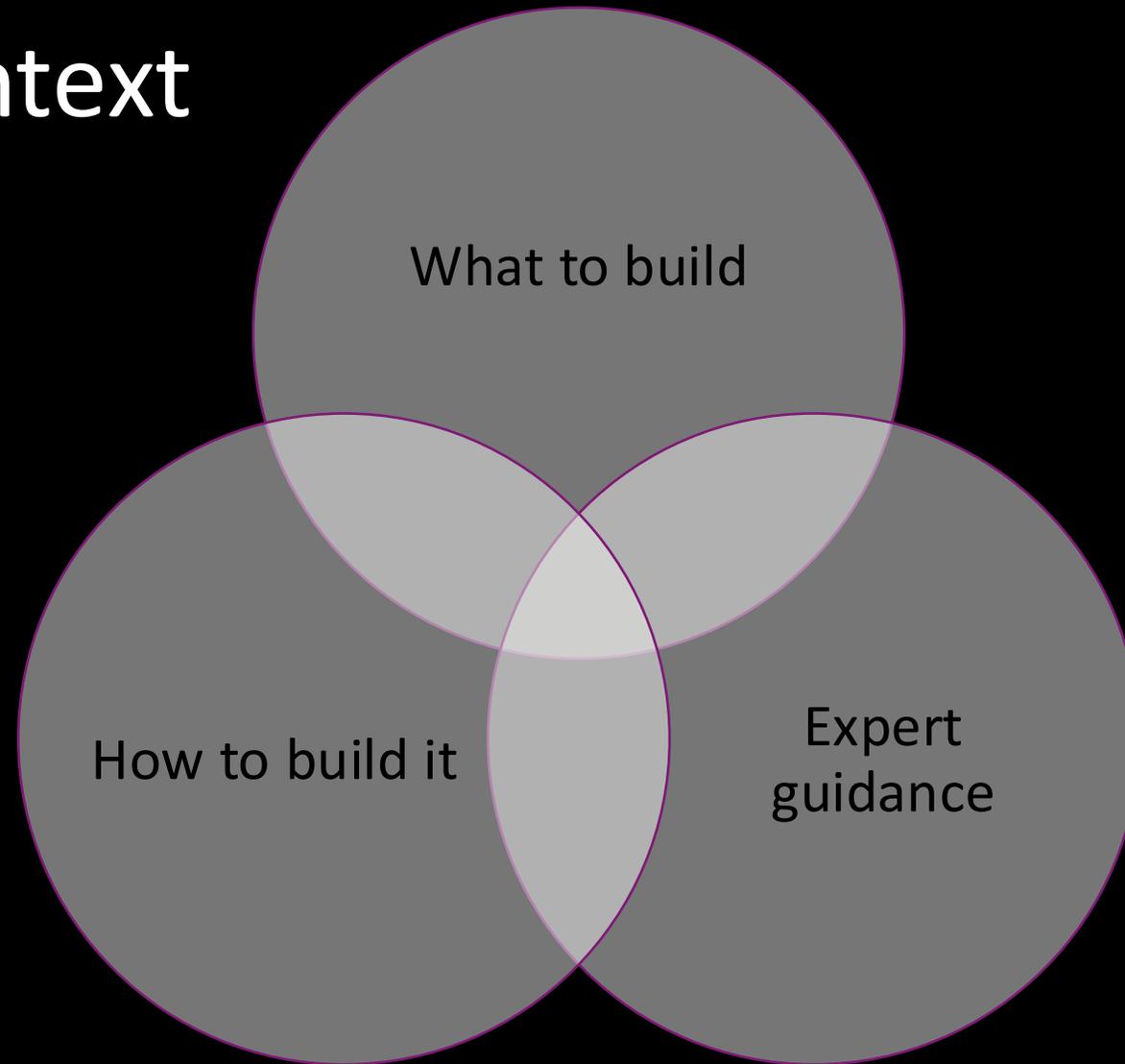
Spec-driven development workflow



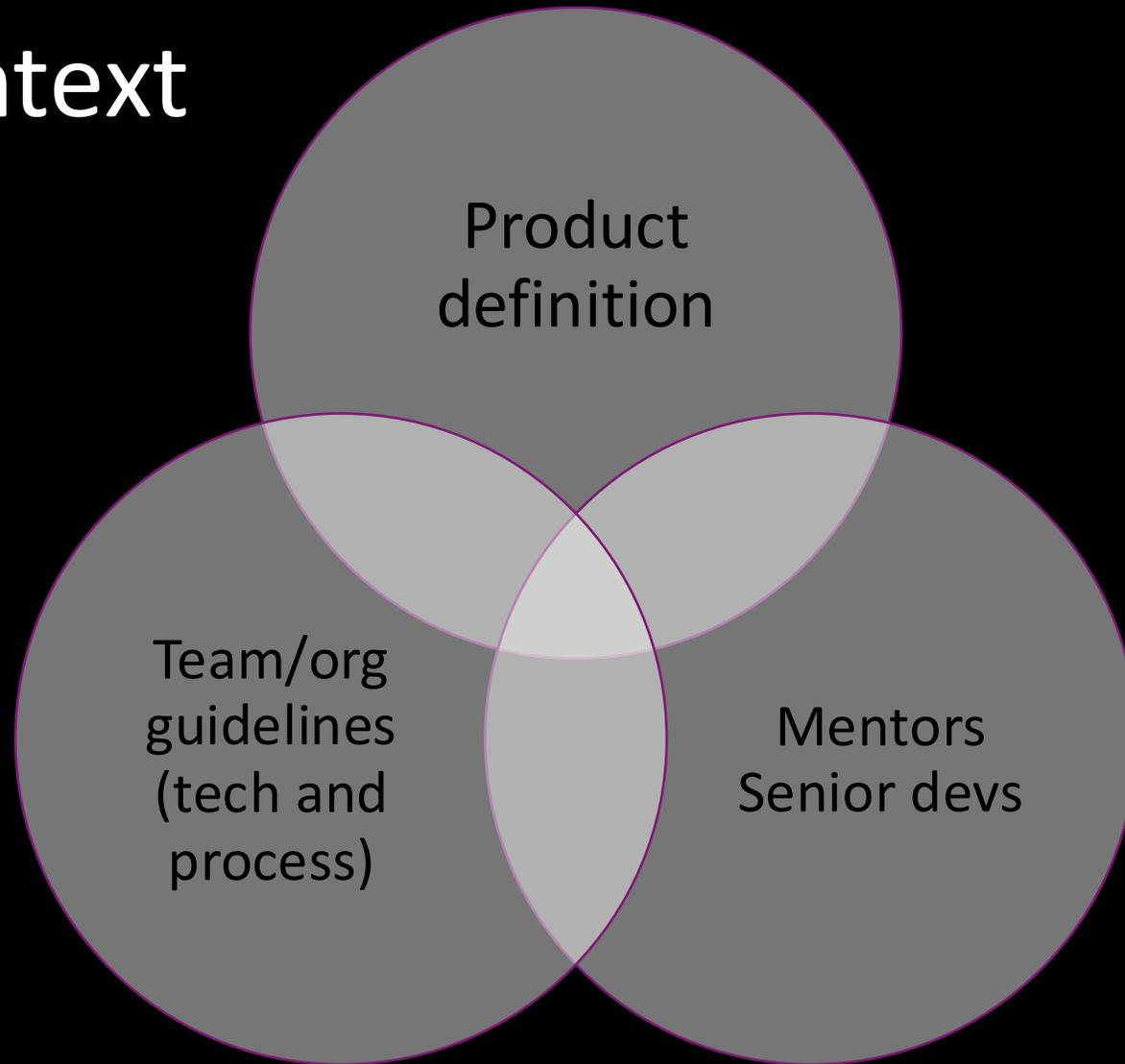
Context

Team onboarding

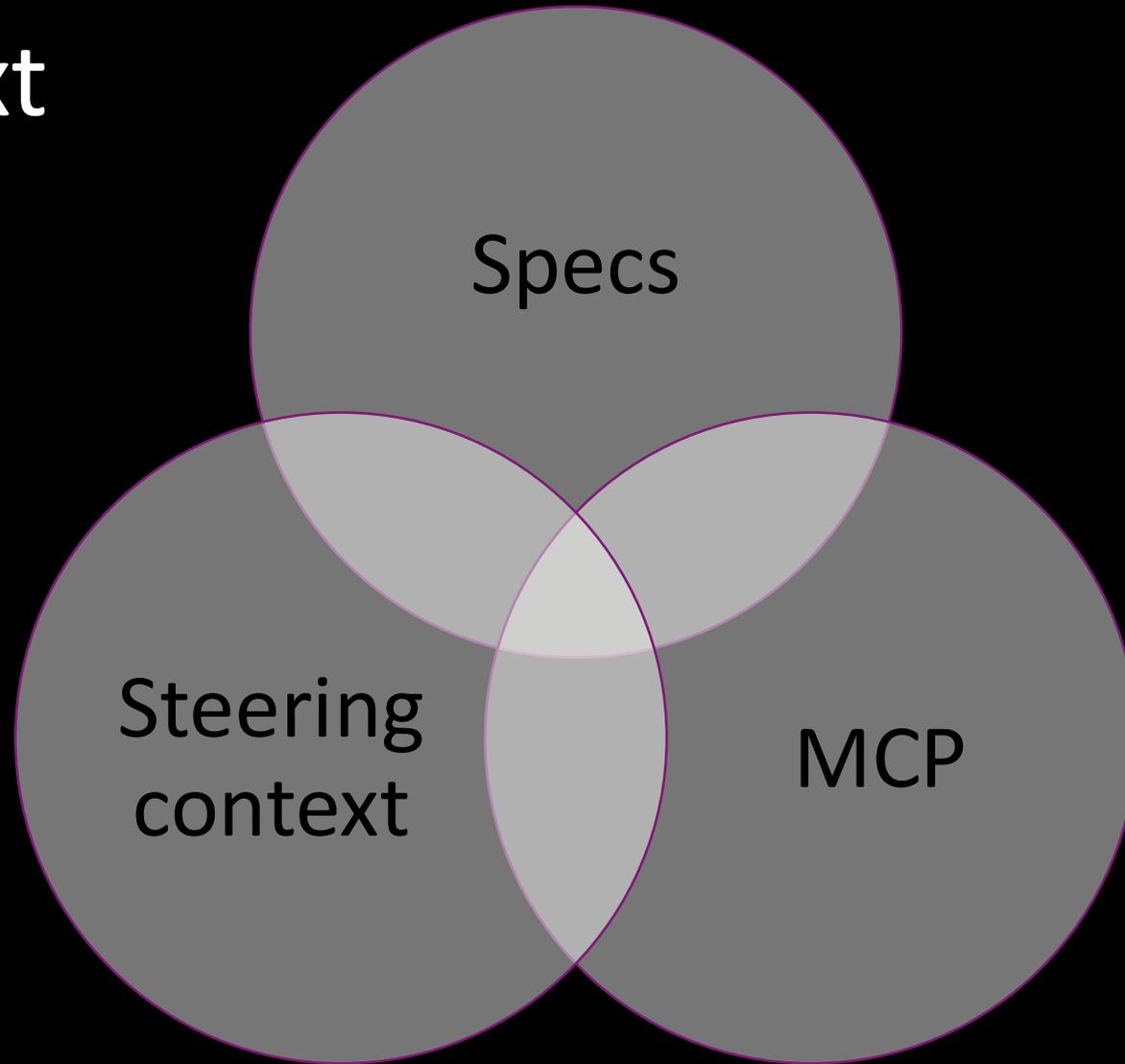
Human context



Human context



SDD context



Introducing Kiro



KIRO

- SPECS
 - weather-forecast-app
- AGENT HOOKS
- AGENT STEERING
 - Workspace
 - mcp
 - product
 - structure
 - tech
- MCP SERVERS
 - fetch Connected (1 tool) ✓
 - awslabs-docs Connected (3 tools) ✓
 - awslabs-core Connected (1 tool) ✓
 - awslabs-api Connected (2 tools) ✓
 - awslabs-knowledge-mcp-server Conn... ✓
 - awslabs-terraform Connected (7 tools) ✓
 - awslabs-serverless Disabled
 - awslabs-diagram Connected (3 tools) ✓
 - awslabs-pricing Connected (8 of 9 tools) ✓



KIRO

- Open chat ⌘ L
- Inline chat ⌘ I
- Show All Commands ⌘ ⌘ P
- Go to File ⌘ P
- Find in Files ⌘ ⌘ F
- Start Debugging F5
- Toggle Terminal ^ ⌘ `
- Toggle Full Screen ^ ⌘ F
- Show Settings ⌘ ,

New Session X

Let's build

Plan, search, or build anything

Vibe

Chat first, then build. Explore ideas and iterate as you discover needs.

Spec

Plan first, then build. Create requirements and design before coding starts.

Great for:

- Thinking through features in-depth
- Projects needing upfront planning
- Building features in a structured way

Ask a question or describe a task...

🗨

Claude Sonnet 4.5 Autopilot



Did you know?

Use `/tangent` or `ctrl + t` (customizable) to start isolated conversations (↵) that don't affect your main chat history

`/help` all commands • `ctrl + j` new lines • `ctrl + s` fuzzy search

🤖 You are chatting with Auto

> H



Demonstration

Serverless weather forecast app

Specs integrated with Github issue tracking for team visibility

Sample weather forecast app Add status update

Backlog Priority board Team items Roadmap My items + New view

Filter by keyword or by field

Backlog	Ready	In progress	In review	Done
1 / 5 Estimate: 0 This item hasn't been started	0 Estimate: 0 This is ready to be picked up	0 / 3 Estimate: 0 This is actively being worked on	0 / 5 Estimate: 0 This item is in review	4 Estimate: 0 This has been completed
<ul style="list-style-type: none">terraform-aws-weather-forecast #8 feat: OpenAPI spec and API contract documentation				<ul style="list-style-type: none">terraform-aws-weather-forecast #7 feat: weather forecast source attributionterraform-aws-weather-forecast #6 Improved background color: shift default gradient from purple to blue-dominantterraform-aws-weather-forecast #9 Optimize agent steering configuration according to official best practicesterraform-aws-weather-forecast #10 feat: Animation that corresponds to the weather forecast

Serverless weather forecast app

Specs

- *Add a feature requirement:*

As an end user I would prefer to see an animation of the forecasted weather for each city. I would also like to see the forecast in plain text, including wind.

- *Results in creation of a new feature spec, integrated with Github issue management*

Leverages the Github MCP Server

From silo development to full team visibility

Serverless weather forecast app

Specs

Add new feature requirement: As an end user I would prefer to see an animation of the forecasted weather for each city. I would also like to see the forecast in plain text, including wind.

 Kiro

 Including Steering Documents `product.md` `specs-issue-workflow.md` `workflow.md`

Based on your description, this sounds like a new feature.

Input required

Is this a new feature or a bugfix?

Build a Feature

Recommended

Implement new functionality or capabilities that don't exist yet

Fix a Bug

Fix something that's broken, crashing, or not working correctly

Cancel

Ask a question or describe a task...



Claude Sonnet 4.6  Autopilot

Serverless weather forecast app

Specs

Input required

What do you want to start with?

Requirements Recommended
Begin by gathering and documenting requirements

Technical Design
Begin with the technical design, then derive requirements from that design

Cancel Submit answer

Requirements

Including Steering Documents `product.md` `specs-issue-workflow.md` `workflow.md`

Searched workspace

- Exploring the frontend source structure to understand existing components
- Checking existing specs to understand the spec directory structure
- Checking if there are already files in the weather-animation-text-forecast spec directory
- Exploring existing frontend components to understand what's already built

Read file(s) `JS WeatherAnimation.js` `JS WeatherTextSummary.js` `JS WeatherCard.js` `JS WeatherDisplay.js` `requirements.md` `design.md` `tasks.md`

Including Steering Documents `frontend.md`

Serverless weather forecast app

Specs integrated with Github issue tracking

Accepted creation of `requirements.md`

The requirements doc is ready. Here's a summary of what's covered:

1. Weather animations — CSS-based animated visuals per city card for 7 weather conditions (`clearsky`, `rain`, `snow`, etc.), with `prefers-reduced-motion` support and graceful fallback for unknown conditions
2. Plain-text summary — human-readable sentence per city including temperature, condition, and wind speed in km/h with cardinal direction (e.g. "from the south-west")
3. Backend extension — expose `windDirection` (degrees) from met.no's `wind_from_direction` field alongside the existing `windSpeed`

Take a look and let me know if anything needs adjusting, or say "looks good" to move on to the design.

Design doc is ready. Here's what it covers:

Architecture — a small data flow extension: Lambda picks up `wind_from_direction` from met.no and passes it as `windDirection` through DynamoDB → API Gateway → `WeatherCard`, which gains two new child components.

Two new frontend components:

- `WeatherAnimation` — pure CSS animated scenes for 7 conditions, `aria-hidden`, `prefers-reduced-motion` support, `cloudy` fallback for unknowns
- `WeatherTextSummary` — sentence-case plain English summary with m/s → km/h conversion and 16-point compass direction labels

Backend change — one-liner addition to `extract_tomorrow_forecast` to read `wind_from_direction`, with `null` fallback if absent.

8 correctness properties covering wind speed conversion, cardinal direction periodicity, summary formatting, animation fallback behaviour, and the backend round-trip — all mapped to specific acceptance criteria.

Testing uses Hypothesis (backend) and fast-check (frontend), both already in the project's dependencies.

Let me know if anything needs tweaking, or say "looks good" to move on to tasks.

All done. Issue #10 has been updated with the finalized requirements, the spec complete comment is posted, `requirements.md` links back to the issue, and `tasks.md` has the close-issue task at the end.

Serverless weather forecast app

Specs integrated with Github issue tracking

Requirement 1: Weather Animation per City Card

User Story: As an end user, I want to see an animation that corresponds to the forecasted weather for each city, so that I can understand the forecast at a glance without reading text.

Acceptance Criteria

1. WHEN a WeatherCard renders with a known `condition`, THE Weather_Animation SHALL display an animation that visually corresponds to that condition (e.g. animated sun rays for `clearsky`, falling drops for `rain`, drifting flakes for `snow`)
2. THE WeatherCard SHALL support distinct animations for the following conditions: `clearsky`, `partlycloudy`, `cloudy`, `rain`, `snow`, `fog`, `thunderstorm`
3. IF the `condition` value is unknown or absent, THEN THE Weather_Animation SHALL display a neutral default animation (e.g. slowly drifting clouds)
4. WHILE the WeatherCard is in a loading state, THE Weather_Animation SHALL NOT be rendered
5. WHILE the WeatherCard is in an error state, THE Weather_Animation SHALL NOT be rendered
6. THE Weather_Animation SHALL be implemented using CSS animations using only `transform` and `opacity` properties to ensure rendering performance
7. WHERE the user has enabled `prefers-reduced-motion`, THE Weather_Animation SHALL reduce or eliminate motion while preserving the static weather visual
8. THE Weather_Animation SHALL be fully visible and correctly sized on viewport widths from 320px to 1440px

As Kiro requirements in EARS format are generated, they are reflected on the Github issue

Serverless weather forecast app

Specs integrated with Github issue tracking

Requirement 2: Plain-Text Weather Summary

User Story: As an end user, I want to see a plain-text description of the forecast for each city including wind information, so that I can quickly understand the conditions in words.

Acceptance Criteria

1. WHEN a WeatherCard renders with valid forecast data, THE Text_Summary SHALL display a human-readable sentence describing the forecast including temperature, condition description, and wind speed
2. THE Text_Summary SHALL display wind speed in km/h, converted from the m/s value provided by the API (1 m/s = 3.6 km/h), rounded to the nearest whole number
3. WHERE wind direction data is available, THE Text_Summary SHALL include a cardinal or intercardinal direction label (e.g. "from the north-west") derived from the `wind_from_direction` degree value
4. IF wind speed data is absent from the API response, THEN THE Text_Summary SHALL display the temperature and condition description without a wind component
5. THE Text_Summary SHALL use sentence case and plain English (e.g. "Partly cloudy with light winds of 12 km/h from the south-west. High of 14°C.")
6. WHILE the WeatherCard is in a loading state, THE Text_Summary SHALL NOT be rendered
7. WHILE the WeatherCard is in an error state, THE Text_Summary SHALL NOT be rendered
8. THE Text_Summary SHALL be readable at all viewport widths from 320px to 1440px without horizontal overflow

Serverless weather forecast app

Specs integrated with Github issue tracking

Requirement 3: Backend Wind Direction Exposure

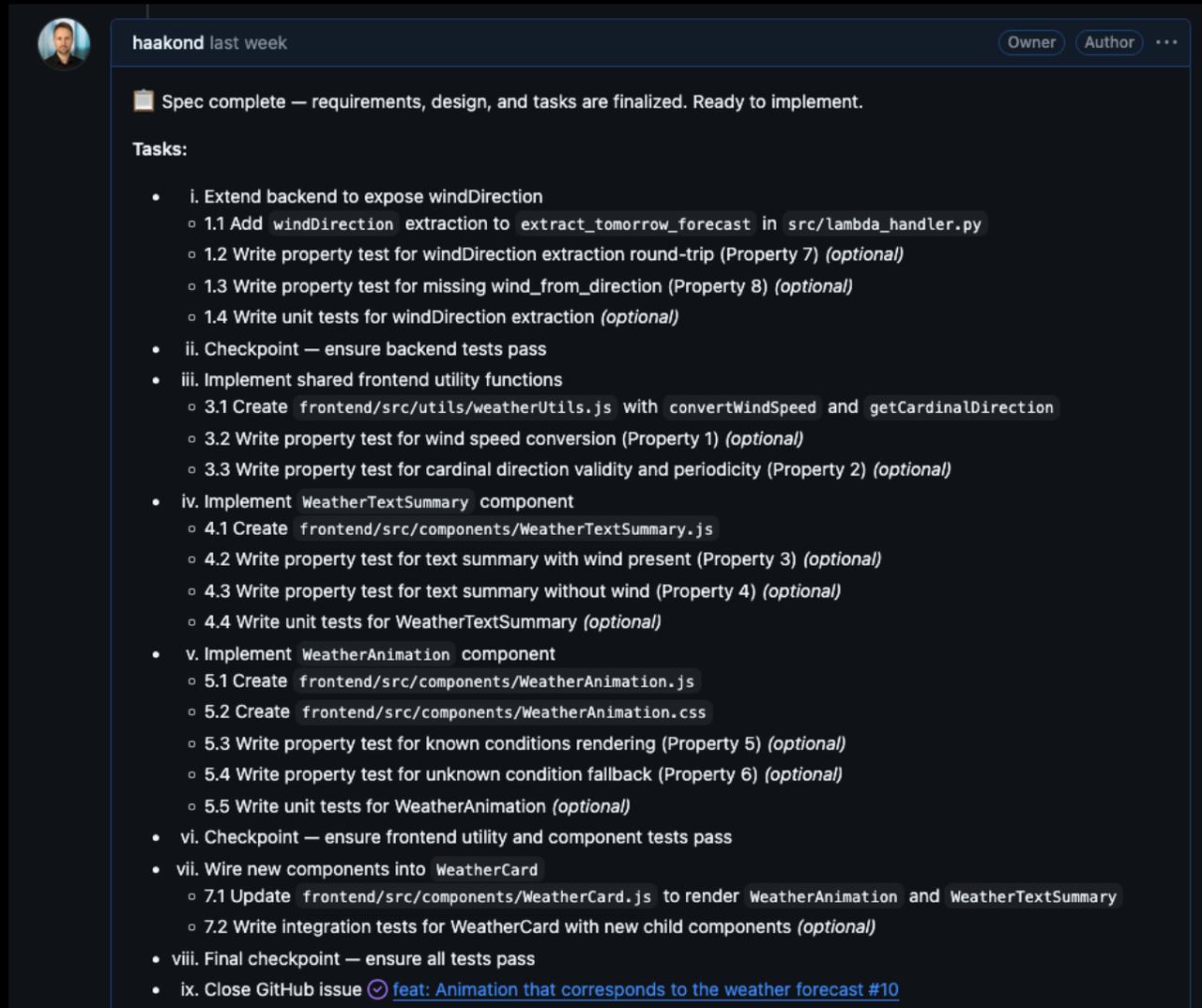
User Story: As a frontend developer, I want the API response to include wind direction, so that the text summary can display a meaningful wind description.

Acceptance Criteria

1. WHEN the Lambda handler extracts tomorrow's forecast from the `met.no` timeseries, THE `Lambda_Handler` SHALL include the `wind_from_direction` value (degrees, 0–360) from the `instant.details` block in the forecast response
2. THE `Lambda_Handler` SHALL expose wind direction as a `windDirection` field (numeric, degrees) alongside the existing `windSpeed` field in the city forecast JSON
3. IF the `wind_from_direction` field is absent from the `met.no` response, THEN THE `Lambda_Handler` SHALL set `windDirection` to `null` in the response without raising an error
4. THE `WeatherForecast` data model SHALL include an optional `wind_direction` field of type `Optional[float]` validated to the range 0–360 inclusive

Serverless weather forecast app

Specs integrated with Github issue tracking



haakond last week Owner Author ...

📄 Spec complete — requirements, design, and tasks are finalized. Ready to implement.

Tasks:

- i. Extend backend to expose windDirection
 - 1.1 Add `windDirection` extraction to `extract_tomorrow_forecast` in `src/lambda_handler.py`
 - 1.2 Write property test for windDirection extraction round-trip (Property 7) *(optional)*
 - 1.3 Write property test for missing `wind_from_direction` (Property 8) *(optional)*
 - 1.4 Write unit tests for windDirection extraction *(optional)*
- ii. Checkpoint — ensure backend tests pass
- iii. Implement shared frontend utility functions
 - 3.1 Create `frontend/src/utils/weatherUtils.js` with `convertWindSpeed` and `getCardinalDirection`
 - 3.2 Write property test for wind speed conversion (Property 1) *(optional)*
 - 3.3 Write property test for cardinal direction validity and periodicity (Property 2) *(optional)*
- iv. Implement `WeatherTextSummary` component
 - 4.1 Create `frontend/src/components/WeatherTextSummary.js`
 - 4.2 Write property test for text summary with wind present (Property 3) *(optional)*
 - 4.3 Write property test for text summary without wind (Property 4) *(optional)*
 - 4.4 Write unit tests for `WeatherTextSummary` *(optional)*
- v. Implement `WeatherAnimation` component
 - 5.1 Create `frontend/src/components/WeatherAnimation.js`
 - 5.2 Create `frontend/src/components/WeatherAnimation.css`
 - 5.3 Write property test for known conditions rendering (Property 5) *(optional)*
 - 5.4 Write property test for unknown condition fallback (Property 6) *(optional)*
 - 5.5 Write unit tests for `WeatherAnimation` *(optional)*
- vi. Checkpoint — ensure frontend utility and component tests pass
- vii. Wire new components into `WeatherCard`
 - 7.1 Update `frontend/src/components/WeatherCard.js` to render `WeatherAnimation` and `WeatherTextSummary`
 - 7.2 Write integration tests for `WeatherCard` with new child components *(optional)*
- viii. Final checkpoint — ensure all tests pass
- ix. Close GitHub issue 🗑️ [feat: Animation that corresponds to the weather forecast #10](#)

When the design is review and approved the Kiro task list is generated, which also is reflected on the Github issue for team visibility

Serverless weather forecast app

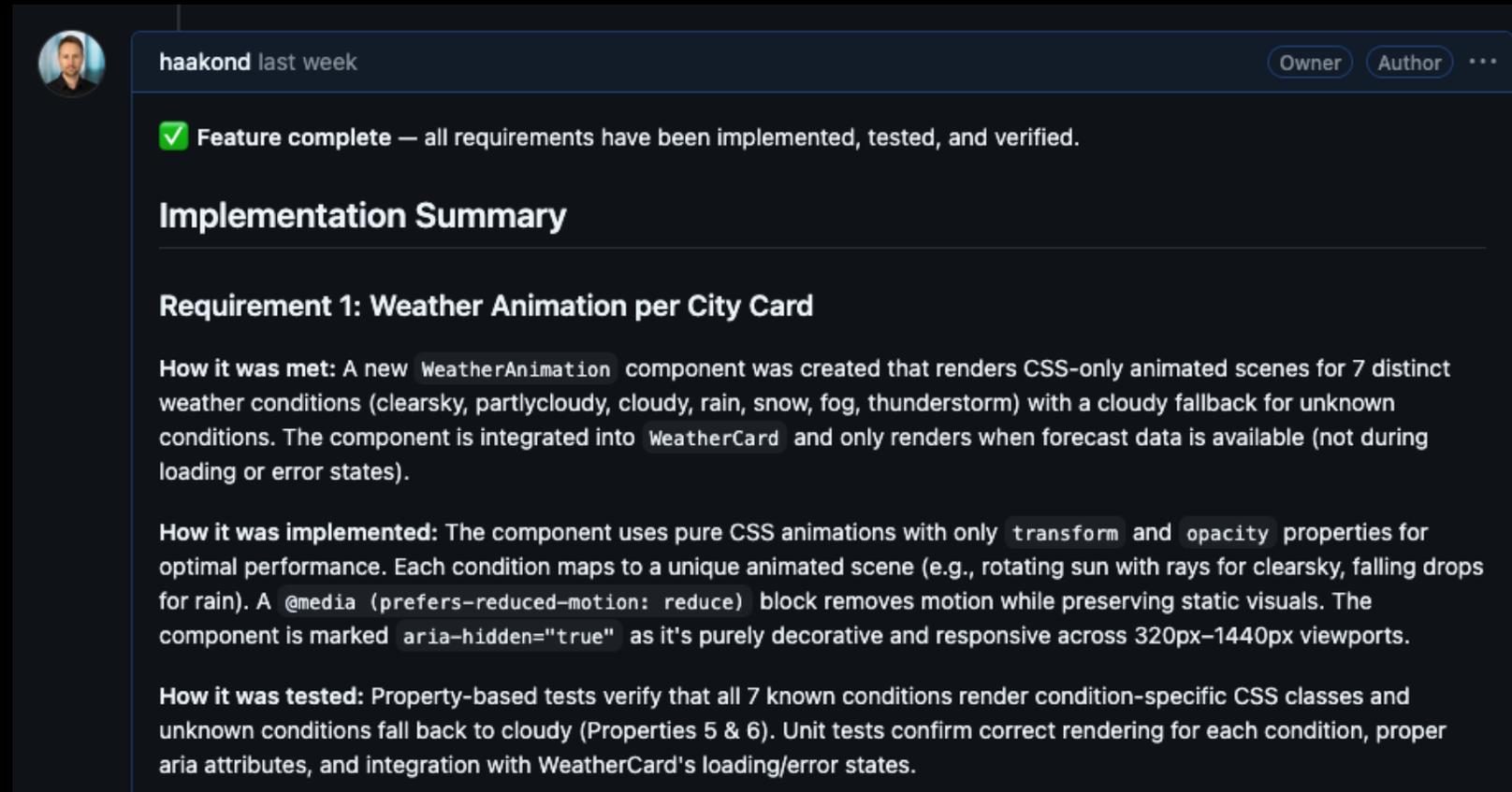
Specs integrated with Github issue tracking

As Kiro tasks are being started, status updates are posted to the corresponding Github issue



Serverless weather forecast app

Specs integrated with Github issue tracking



haakond last week Owner Author ...

✅ **Feature complete** — all requirements have been implemented, tested, and verified.

Implementation Summary

Requirement 1: Weather Animation per City Card

How it was met: A new `WeatherAnimation` component was created that renders CSS-only animated scenes for 7 distinct weather conditions (clearsky, partlycloudy, cloudy, rain, snow, fog, thunderstorm) with a cloudy fallback for unknown conditions. The component is integrated into `WeatherCard` and only renders when forecast data is available (not during loading or error states).

How it was implemented: The component uses pure CSS animations with only `transform` and `opacity` properties for optimal performance. Each condition maps to a unique animated scene (e.g., rotating sun with rays for clearsky, falling drops for rain). A `@media (prefers-reduced-motion: reduce)` block removes motion while preserving static visuals. The component is marked `aria-hidden="true"` as it's purely decorative and responsive across 320px–1440px viewports.

How it was tested: Property-based tests verify that all 7 known conditions render condition-specific CSS classes and unknown conditions fall back to cloudy (Properties 5 & 6). Unit tests confirm correct rendering for each condition, proper aria attributes, and integration with `WeatherCard`'s loading/error states.

As tasks are completed, a summary is added to the Github issue, for team visibility.

For each requirement:

- How it was met
- How it was implemented
- How it was tested

Serverless weather forecast app

Specs integrated with Github issue tracking

Requirement 2: Plain-Text Weather Summary

How it was met: A new `WeatherTextSummary` component generates human-readable forecast sentences including temperature, condition description, and wind information (speed in km/h with cardinal direction). The component handles missing wind data gracefully and only renders when forecast data is available.

How it was implemented: Two utility functions (`convertWindSpeed` and `getCardinalDirection`) handle unit conversion (m/s \rightarrow km/h) and degree-to-compass mapping (16-point compass with 22.5° sectors). The component builds sentence-case output with three variants: full wind+direction, wind-only, or no wind. Text is rendered in a responsive `<p>` element that prevents overflow on all viewport sizes.

How it was tested: Property-based tests verify correct unit conversion (Property 1), valid periodic cardinal direction mapping (Property 2), presence of required fields when wind is present (Property 3), and proper omission of wind when absent (Property 4). Unit tests cover all sentence variants, null forecast handling, and integration with `WeatherCard` states.

As tasks are completed, a summary is added to the Github issue, for team visibility.

For each requirement:

- How it was met
- How it was implemented
- How it was tested

Serverless weather forecast app

Specs integrated with Github issue tracking

Requirement 3: Backend Wind Direction Exposure

How it was met: The Lambda handler's `extract_tomorrow_forecast` function was extended to extract `wind_from_direction` from the met.no API's `instant.details` block and expose it as a `windDirection` field in the forecast JSON response alongside the existing `windSpeed` field.

How it was implemented: A single line was added to read `instant_data.get("wind_from_direction", None)` and include it in the `forecast_data` dictionary. When the field is absent from the met.no response, `windDirection` is set to `null` without raising errors, maintaining backward compatibility.

How it was tested: Property-based tests verify round-trip extraction of wind direction values (Property 7) and correct null handling when the field is absent (Property 8). Unit tests confirm the field is present in the output structure, correctly populated when available, and null when missing.

All 8 correctness properties passed with 100+ test iterations each. Both backend (Python/pytest/Hypothesis) and frontend (Jest/fast-check) test suites are green.



 **haakond** closed this as completed last week

 **github-project-automation** moved this from Backlog to Done in  Sample weather forecast app last week

 **haakond** added 2 commits that reference this issue last week

docs: link issue [#10](#) and add close-issue task to weather-animation-te. 

f98a5f5

feat: add weather animation and text forecast with wind direction sup. 

8a9fc53

As tasks are completed, a summary is added to the Github issue, for team visibility.

For each requirement:

- How it was met
- How it was implemented
- How it was tested

Serverless weather forecast app

Before



Serverless weather forecast app

After



Wrap-up

Spec-Driven Development is rapidly becoming the
norm for
AI assisted software development

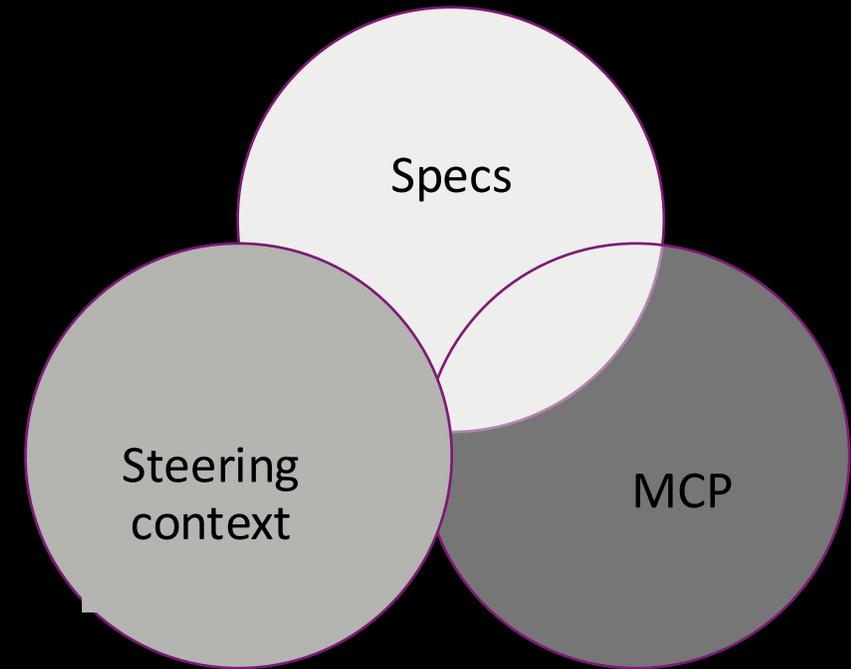
Spec-driven capabilities across AI coding tools

From opinionated to modular

	Kiro	Cursor	Claude Code	VS Code (Copilot)	Windsurf
Approach	Native spec workflow	Plan Mode (bolt-on)	Plan Mode (bolt-on)	Plan agent + Spec Kit (external add-on)	Planning Mode (conversational)
SDLC phases	<ol style="list-style-type: none">RequirementsDesignTasks	<ol style="list-style-type: none">PromptPlanExecute	<ol style="list-style-type: none">PromptPlanExecute	<ol style="list-style-type: none">DiscoverAlignDesignRefine	<ol style="list-style-type: none">PromptStepsExecute
Persistent artifacts	requirements.md design.md tasks.md - versioned in repo	Session-scoped plan	User-managed files	External tool generates files	None
Task tracking	Built-in status tracking & incremental execution	Checkboxes in plan UI	Manual	Checkboxes (session only)	Step progress only
AGENTS.md / SKILL.md	Reads AGENTS.md; own steering files (.kiro/steering/); Powers bundles SKILL.md with MCP servers	Reads AGENTS.md; reads SKILL.md	Reads AGENTS.md; native SKILL.md support (plugin system)	Reads AGENTS.md; SKILL.md via agent plugins	Reads AGENTS.md; reads SKILL.md

Best practices: Specs

- 1 Spec = 1 User Story

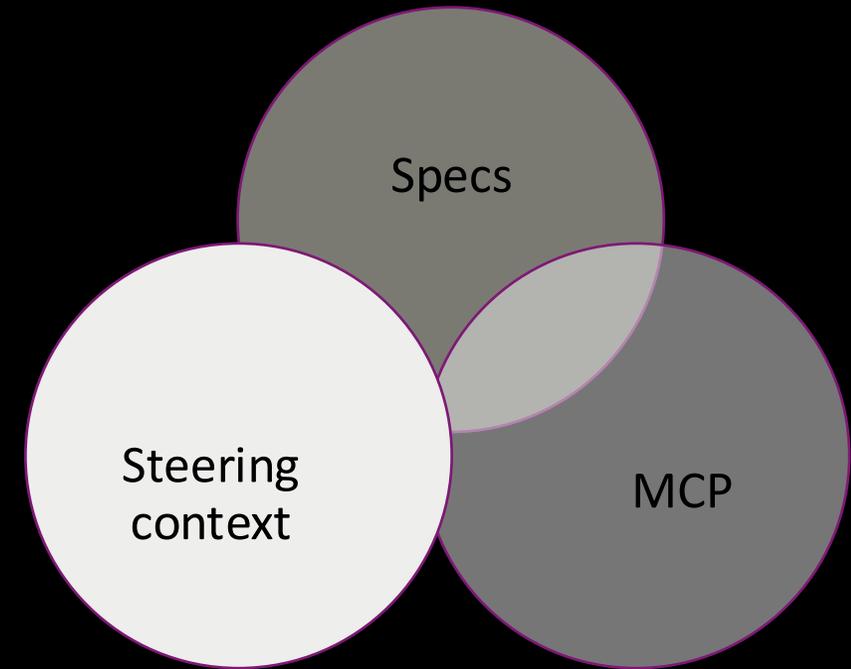


Optimize context to reduce token usage



Best practices: Steering Context

Agent Steering/Skill/Plan



Each definition should address a specific capability:

- "PDF form filling"
- "Excel data analysis"
- "Git commit messages"
- Python
- Frontend
- "AWS tech stack definition"
- "Team tenets"
- "Product/project goals"

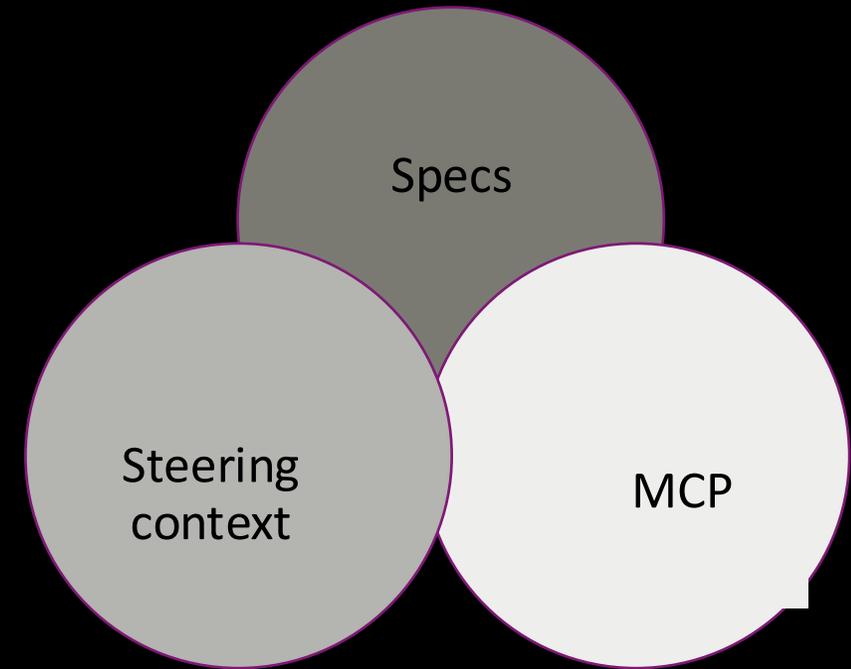
Too broad:

- "Document processing" (split into separate definitions)
- "Data tools" (split by data type or operation)
- "Development best practices"

Best practices: MCP Servers

Expert guidance, Model Context Protocol

- Each MCP server should address a specific capability for your project scope
- Stay below max 50 tools, disable/remove unused entries



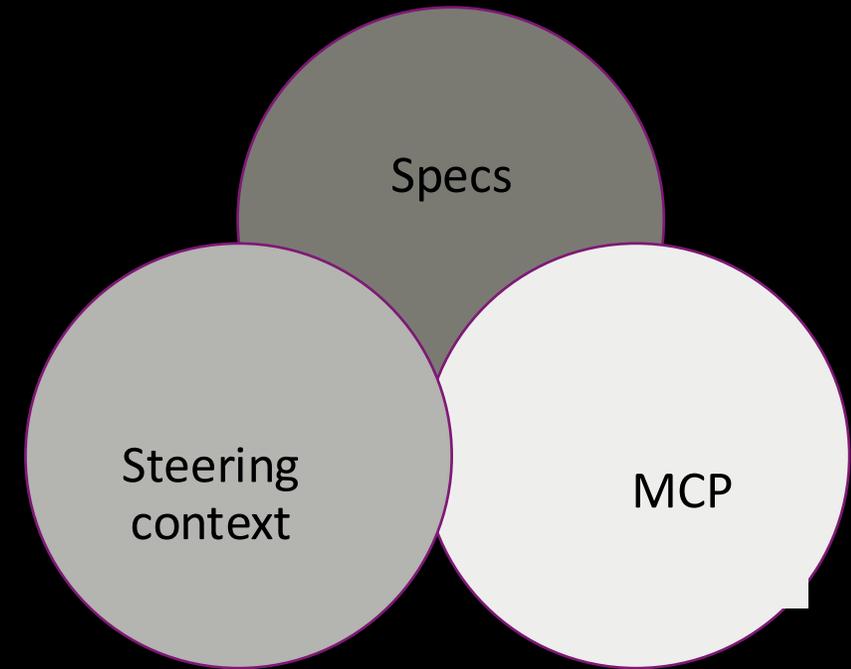
 You have 68 MCP tools enabled. We recommend disabling servers or tools to keep this below 50, as too many tools lead to degraded agent tool selection and high token usage consuming significant context. You can disable servers and tools from the MCP Servers view.  

Source: kiroAgent

[Show MCP Servers](#)

Best practices: MCP Servers

Expert guidance, Model Context Protocol



- Each MCP server should address a specific capability for your project scope
- Stay below max 50 tools, disable/remove unused entries
- MCP Server registries:

<https://kiro.dev/docs/mcp/servers/>

<https://github.com/mcp>

GitHub	+ Add to Kiro	Interact with GitHub repositories, issues, and pull requests.
GitLab	+ Add to Kiro	GitLab MCP server allows you to plan, track, and manage issues, merge requests, and pipelines from Kiro.
Kubernetes	+ Add to Kiro	Interact with Kubernetes clusters. Requires Node installed

Best practices: Subagents

Divide, conquer and optimize context usage

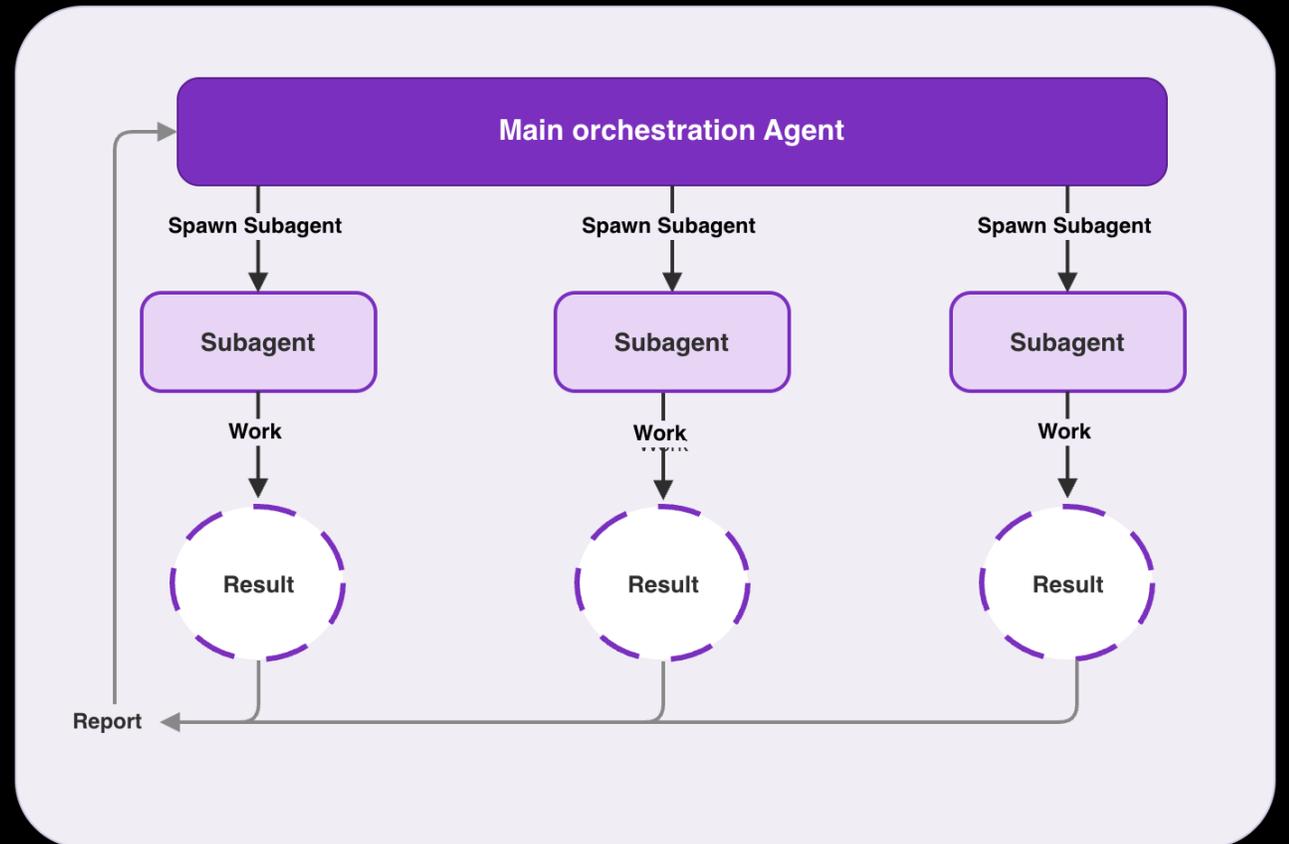
- Kiro's main agent can spawn specialized subagents that work autonomously and in parallel, each with their own tools and focused context.
- Summarized results flow back to the main agent, which orchestrates the final outcome.

Parallelization - multiple domains worked simultaneously

Domain specialists - each subagent brings focused expertise (frontend, backend, infra, security, docs)

Context stays lean - orchestrator holds the big picture; subagents load only what they need. No wasted tokens, no diluted attention.

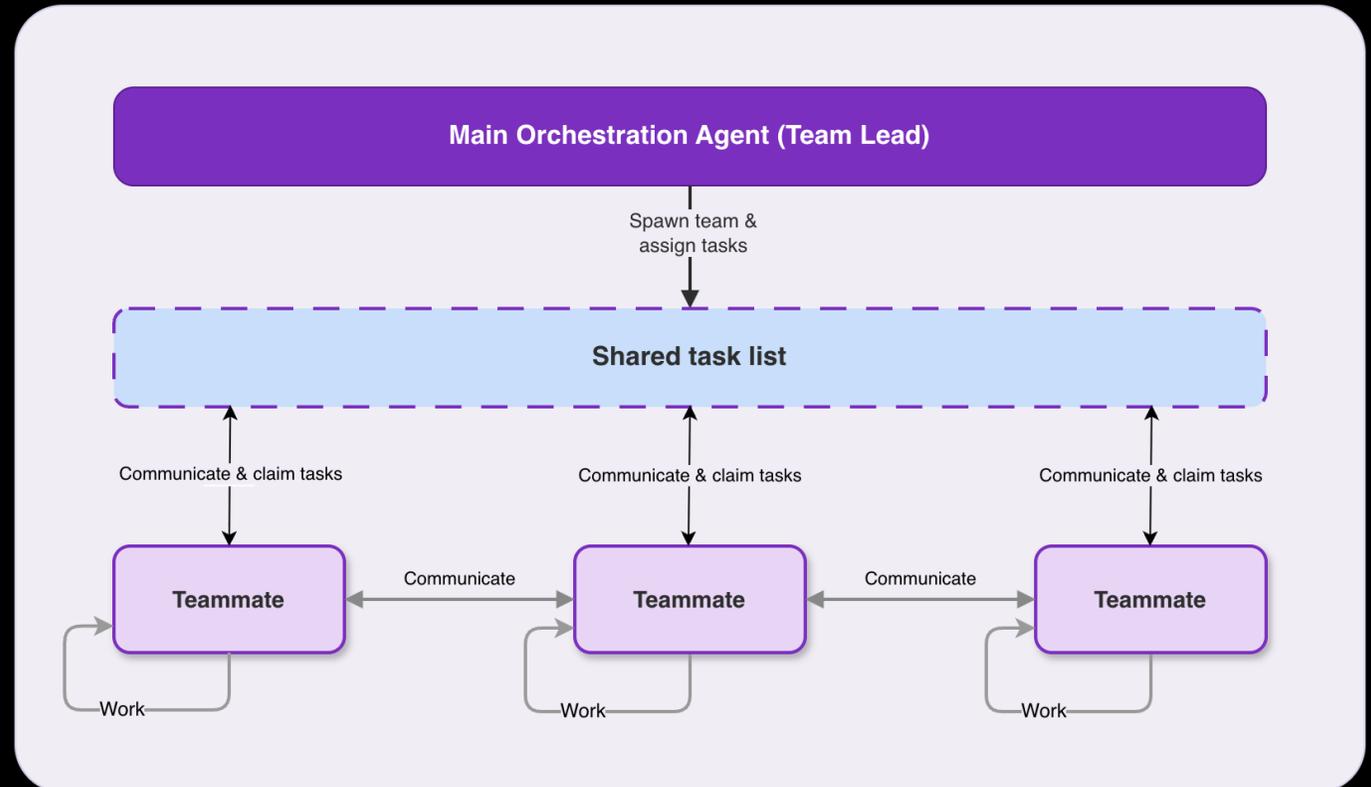
Scales naturally - simple tasks stay simple; complex work fans out across agents



What's coming: Agent Teams

Parallelize even more, increased context usage

- Agent teams increases token usage because of coordination
- Consider implementation time vs. cost of token usage
- Already supported in Claude Code
- Use subagents when you need quick, focused workers that report back.
- Use agent teams when teammates need to share findings, challenge each other, and coordinate on their own.



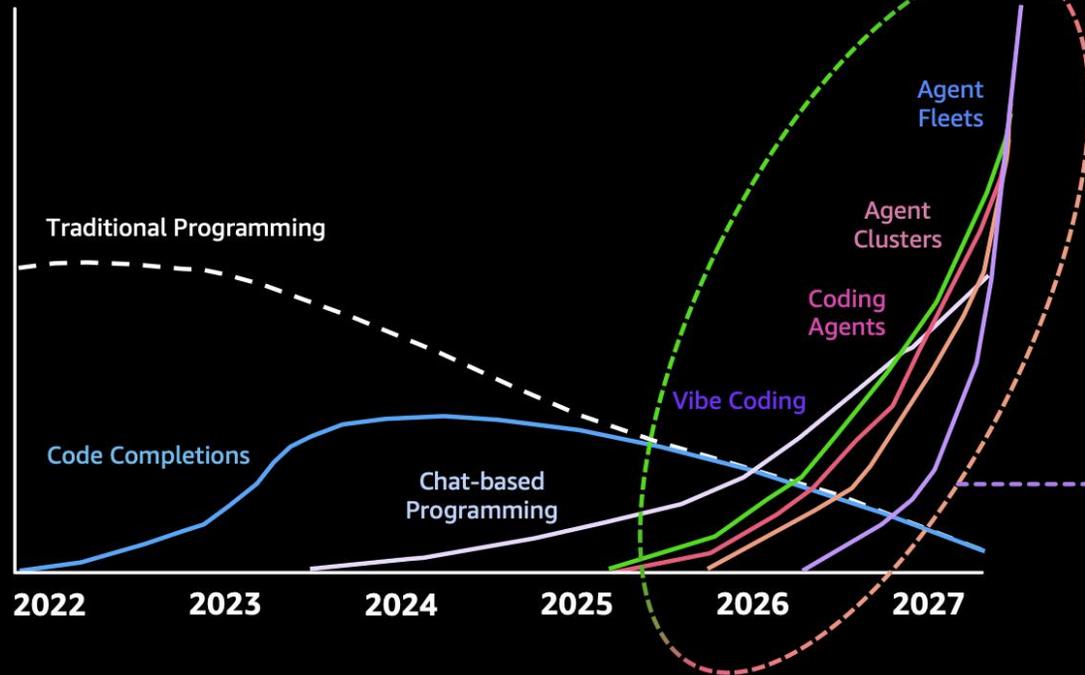
What does the future look like?

Code will no longer be the source of truth

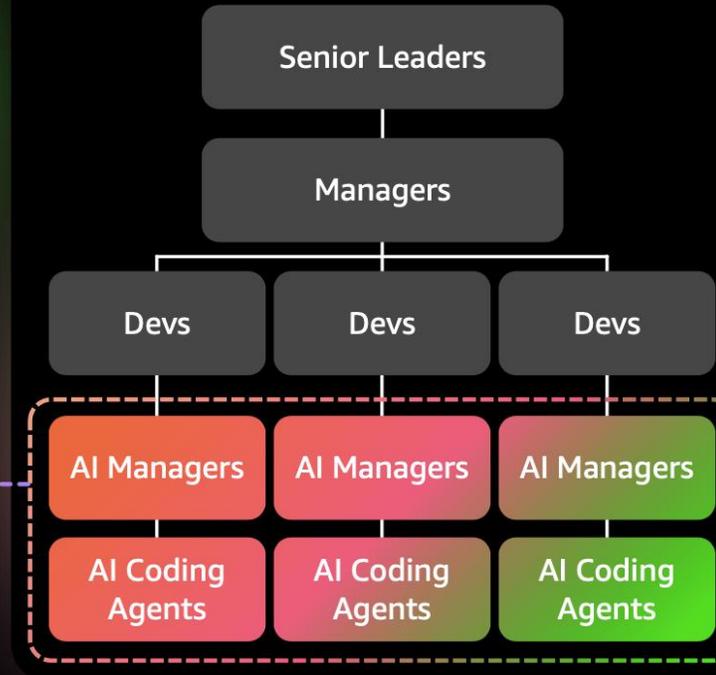
Intent becomes the source of truth

Software development is changing

Evolution of AI coding strategies



The software development team



Get started with Spec-Driven Development

- <https://kiro.dev/docs/guides/learn-by-playing/>
- <https://aws.amazon.com/blogs/devops/ai-driven-development-life-cycle/>
- <https://hedrange.com/2025/08/11/how-to-use-kiro-for-ai-assisted-spec-driven-development/>
- <https://github.com/obra/superpowers>

Kiro.dev



KIRO



sopra  steria

The logo for Sopra Steria, featuring the word 'sopra' in white, a stylized 'S' logo in red and yellow, and the word 'steria' in white.