[ES] [Tech-debt] OpsPortal - Spec Driven Development

Speckit Developer Documentation

Last Updated: November 10, 2025

Version: 2.0

Audience: Developers, Product Managers, Technical Leads

Workflow: Constitution → Specify → Clarify → Dive-in → Plan → Tasks → Analyze → Implement

---

Table of Contents

Introduction

> **Info:** Speckit is a **specification-driven development workflow** that transforms natural language feature descriptions into fully-planned, implementation-ready development tasks.

Why Speckit?

Traditional Problems	Speckit Solutions
Requirements scattered across documents, chats, meetings	**Single source of truth**: All specs, plans, and tasks in `specs/` directory
Implementation starts before design complete	**Progressive refinement**: Each stage builds on previous work
Ambiguities discovered mid-development	**Explicit clarifications**: Caught early and resolved systematically

No traceability between requirements and code	**Full traceability**: Requirements → Plan → Tasks → Code
Teams unsure if they're building the right thing	**Constitution-backed**: Features validated against project principles

---

Here is the new section "When NOT to Use Speckit" formatted for your documentation. I recommend placing it after the "Best Practices" section and before "Examples & Use Cases" for maximum clarity.

When NOT to Use Speckit

❌ DO NOT use Speckit for:

Scenario	Why It's Inappropriate	Better Alternative
Bug fixes	No new feature/requirements needed	Direct fix with git commit
Hotfixes	Time-critical, needs immediate deployment	Emergency patch process
Trivial UI tweaks	Change button color, adjust spacing	Direct edit with code review
Copy/text updates	Change wording, fix typos	Direct content update
Dependency updates	Upgrade package versions	Standard dependency management
Configuration changes	Update environment variables, API keys	Config management tools
Refactoring existing code	Improving code quality without new functionality	Dedicated refactoring workflow
Emergency investigations	Debugging production issues	Incident response process
Solo hobby projects	Personal learning, no team coordination needed	Freestyle development
One-off admin scripts	Single-use data migration or cleanup	Script with inline comments

Documentation-only updates	README improvements, API docs	Direct documentation editing

Speckit is designed for:

- ✅ **New features** with user-facing functionality
- ✅ **Medium to complex** changes (>1 day of work)
- ✅ **Team collaboration** requiring shared understanding
- ✅ **Long-term maintenance** where traceability matters
- ✅ **Multiple stakeholders** needing alignment
- ✅ **API integrations** requiring contracts and error handling

What is Speckit?

Speckit is a collection of **8 interconnected AI commands** that guide you through a complete feature development lifecycle:

`Constitution → Specify → Clarify → Dive-in → Plan → Tasks → Analyze → Implement`

Command Overview

Command	Purpose	Output
`/speckit.constitution`	Establish project principles	`constitution.md`
`/speckit.specify`	Create feature specification	`spec.md`, `requirements.md`, `progress.md`
`/speckit.clarify`	Resolve ambiguities (max 5 questions)	Updated `spec.md`
`/speckit.dive-in`	Document backend APIs	`dive-in.md`
`/speckit.plan`	Design technical architecture	`plan.md`, `research.md`, `data-model.md`, `contracts/`
`/speckit.tasks`	Generate implementation tasks	`tasks.md`

`/speckit.analyze`	Validate consistency (read-only)	Analysis report
`/speckit.implement`	Execute tasks and build feature	Source code, updated `tasks.md`

Core Concepts

Feature Branches

Every feature gets a numbered branch: `###-short-descriptive-name`

Examples: `001-user-authentication`, `002-payment-processing`

Artifacts

Each feature produces a structured set of documents:

```
specs/001-user-authentication/
├── spec.md
├── dive-in.md
├── plan.md
├── research.md
├── data-model.md
├── quickstart.md
├── tasks.md
├── contracts/
│   ├── auth-api.md
│   └── user-api.md
└── checklists/
    ├── requirements.md
    ├── progress.md
    ├── ux.md
    └── security.md
```

Constitution

The **project constitution** (`.specify/memory/constitution.md`) defines:

- Architectural principles
- Quality standards
- Governance processes

> **Warning:** All features MUST align with constitutional principles.

Progress Tracking

Each feature tracks progress through workflow stages:

Stage	Status	Completed
1. Constitution	✓ Complete	2025-11-03
2. Specify	✓ Complete	2025-11-04
3. Clarify	In Progress	• 1

...	...	...

Installation & Setup

Prerequisites

- AI IDE (Cursor, GitHub Copilot, Claude Code, etc.)
- Git repository initialized
- Project structure with `prompts/` and `.specify/` folders
- Atlassian MCP or Dive-in content in markdown file

Step 1: Install Prompts

```
./prompts/setup_prompts.sh
# Or specify your IDE
./prompts/setup_prompts.sh cursor
```

Step 2: Verify Installation

- Restart your IDE
- Type `/speckit.` in chat or command palette to see suggestions

Step 3: Verify Project Structure

```
ls -la .specify/
ls -la .specify/memory/
ls -la .specify/templates/
```

Step 4: Initialize Constitution

```
/speckit.constitution
```

The Workflow

Overview: 8 Stages

```
graph LR
    A[Constitution] --> B[Specify]
    B --> C[Clarify]
    C --> D[Dive-in]
    D --> E[Plan]
    E --> F[Tasks]
    F --> G[Analyze]
    G --> H[Implement]
```

Stage	What It Does	Required?	When to Skip
1. Constitution	Define project principles	Yes (once per project)	Constitution exists
2. Specify	Create feature specification	Yes (every feature)	Never
...	...	...	...

Command Reference

1. Constitution: `/speckit.constitution`

- Establish or update project architectural principles.
- Output: `.specify/memory/constitution.md`, progress report.

2. Specify: `/speckit.specify`

- Create a business-focused feature specification.
- Output: `spec.md`, requirements checklist, progress.

3. Clarify: `/speckit.clarify`

- Identify and resolve ambiguities in the specification.
- Output: Updated `spec.md`, clarifications, progress.

4. Dive-in: `/speckit.dive-in`

- Create backend API specification.
- Output: `dive-in.md`, frontend checklist, progress.

5. Plan: `/speckit.plan`

- Generate technical implementation plan.
- Output: `plan.md`, `research.md`, `data-model.md`, contracts, progress.

6. Tasks: `/speckit.tasks`

- Generate actionable, dependency-ordered implementation tasks.
- Output: `tasks.md`, progress.

7. Analyze: `/speckit.analyze`

- Perform read-only consistency and quality analysis.
- Output: Analysis report, progress.

8. Implement: `/speckit.implement`

- Execute implementation plan by processing all tasks.
- Output: Updated `tasks.md`, code, progress.

Best Practices

- Always start with Constitution
- Use Clarify liberally
- Run Analyze before Implement
- Implement incrementally (MVP first)
- Complete User Story 0 + 1 first

Examples & Use Cases

Example 1: Simple Feature (Read-Only UI)

Example 2: Complex Feature (CRUD with Backend)

Example 3: Framework/Infrastructure Feature

Troubleshooting

- **Commands not appearing:** Restart IDE, verify prompt files, check IDE version.
- **Setup script fails:** Specify IDE explicitly, check project root, verify environment variables.
- **Workflow issues:** Provide more detail in feature description, accept default answers, clarify refactor intent in spec.

Advanced Topics

- **Custom Checklists:** `/speckit.checklist`
- **Multi-Feature Projects:** Sequential or parallel strategies
- **Constitution Evolution:** Propose, update, review, and version principles

---

Quick Reference Card

```
/speckit.constitution    Setup project principles
/speckit.specify         Create feature spec (WHAT)
/speckit.clarify         Resolve ambiguities (max 5 Q's)
/speckit.dive-in         Document backend APIs
/speckit.plan            Design architecture (HOW)
/speckit.tasks           Generate implementation tasks
/speckit.analyze         Validate consistency (read-only)
/speckit.implement       Execute tasks and build feature
/speckit.checklist       Create quality checklist
```

Summary

Speckit transforms feature development from ad-hoc to systematic:

Before Speckit	With Speckit
❌ Requirements scattered	✅ Single source of truth
❌ Implementation before design	✅ Progressive refinement
❌ Late ambiguity discovery	✅ Early resolution
❌ No traceability	✅ Full traceability
❌ Inconsistent quality	✅ Constitution-backed quality

Key Workflows:

- **Quick**: `constitution → specify → plan → tasks → implement`
- **Recommended**: `constitution → specify → clarify → plan → tasks → analyze → implement`
- **Complete**: All 8 stages (best for complex features)

Best Practices:

- ✅ Start with constitution
- ✅ Focus on WHAT (spec) before HOW (plan)
- ✅ Resolve ambiguities early (clarify)
- ✅ Validate before implementing (analyze)
- ✅ Implement incrementally (MVP first)
- ✅ Default to NEW features (not refactors)
- ✅ Make user stories independent

Reference:

Local AI Code Review by Toan Trieu

GitHub - github/spec-kit: 💫 Toolkit to help you get started with Spec-Driven Development

GitHub - sooperset/mcp-atlassian: MCP server for Atlassian tools (Confluence, Jira)