Specification-Driven Workflow for Claude Code

A practical guide for managing Claude Code projects using a living specification document instead of conversational chat

Author: Gyre Research (team@gyreresearch.com)

Date: 2026-06-11 | Developer Tooling & AI Workflow

Why This Approach Works

The Problem: Context Window Limits

Claude operates within a context window — the text visible during any single interaction. In long chat sessions, earlier messages eventually scroll out of this window. When that happens, Claude Code loses access to prior decisions, requirements, and Q&A. This manifests as:

- Asking questions you already answered
- Contradicting earlier architectural decisions
- Forgetting project-specific constraints
- Requiring repeated explanations

The Solution: A Living Specification Document

Instead of scattering project knowledge across chat messages, consolidate everything into a single SPEC.md file that Claude Code reads at the start of each session. This ensures:

BENEFIT	HOW IT WORKS
Consistent context	Every session starts with the full picture — nothing forgotten
No repeated explanations	Decisions captured once, referenced forever
Clear audit trail	See how requirements evolved and why
Productive sessions	Jump straight to work instead of re-establishing context

A Note on Token Usage

This approach does not necessarily reduce total tokens — you are still providing the same information. However, it restructures token usage beneficially:

- Eliminates redundant clarification conversations
- Removes "remind me what we decided about X" exchanges
- Makes each session more productive per token spent

The real wins are consistency, reduced rework, and better outcomes.

The Specification Document

Create a file named SPEC.md in your project root. Structure it as follows:

```
# Project Name - Specification

**Version:** YYYY-MM-DD Build:N
**Owner:** Your Name
**Status:** Draft | In Progress | Complete

---
## 1. Overview
Brief description of what this project does and why it exists.

---
## 2. Requirements

### 2.1 Functional Requirements
- FR-1: [Requirement description]
- FR-2: [Requirement description]

### 2.2 Non-Functional Requirements
- NFR-1: Must run on Rocky Linux 9
- NFR-2: Must follow Gyre coding standards

### 2.3 Out of Scope
```

```
- [Feature deferred to future phase]

---
## 3. Q&A Log

**Q1 (Claude):** How should the system handle API rate limits?
**A1 (Owner):** Retry with exponential backoff, max 3 attempts, then skip.

---
## 4. Decisions
| ID | Decision | Rationale | Date |
| D-1 | Use retry with backoff | Resilience over speed | 2025-01-29 |

---
## 5. Current Status
- [x] Requirements gathered
- [ ] Main ETL script (60% complete)

---
## 6. Technical Notes
- Endpoint: https://api.example.com/v2/data
- Rate limit: 100 requests/minute

---
## Change Log
| 2025-01-29 | 1 | Your Name | Initial specification |
```

The Workflow

Starting a New Project

- Create SPEC.md with your initial requirements (Sections 1–2 minimum).
- First Claude Code session: read SPEC.md and ask clarifying questions. Append questions to Section 3 (Q&A Log).
- Answer by editing the Q&A section directly in the file.

- Next prompt: read the updated SPEC.md and proceed with implementation. Update Section 5 as you work.

Continuing an Existing Project

Each new session, start with:

```
Read SPEC.md for full context, then continue with [specific task].
```

This single instruction provides everything Claude Code needs — requirements, prior Q&A, decisions, and current status.

When Questions Arise Mid-Session

If Claude Code needs clarification during implementation:

- Append the question to Section 3
- Answer in the document
- Claude re-reads and continues

This keeps all Q&A in one place rather than scattered through chat.

Practical Tips

Keep the Spec Updated

After each significant session, ensure: new decisions are captured in Section 4, status reflects actual progress in Section 5, and any new constraints or learnings are in Section 6.

team@gyreresearch.com ▪ gyreresearch.com ▪ (800) 933-4731

Use Clear Requirement IDs

Reference requirements by ID (FR-1, NFR-2) in code comments and commit messages. This creates traceability.

Don't Over-Specify Initially

Start with high-level requirements. Let the Q&A process surface the details that actually matter. Over-specifying upfront often means specifying the wrong things.

Version the Spec

Update the version (YYYY-MM-DD Build:N) when making significant changes. This helps track when decisions were made.

Split Large Projects

If a project grows beyond approximately 500 lines in the spec, consider splitting into SPEC-overview.md for high-level architecture and decisions, and separate SPEC-module files for detailed requirements per module.

Example Session Flow

Session 1 — Project Kickoff

You: "I have created SPEC.md with initial requirements for a price data ETL. Read it and ask clarifying questions."

Claude: Reads spec, appends questions to Q&A section.

You: Answer questions directly in SPEC.md.

Session 2 — Implementation

You: "Read SPEC.md and implement the main ETL script."

Claude: Reads spec (sees all prior Q&A), implements script, updates status section.

team@gyreresearch.com ▪ gyreresearch.com ▪ (800) 933-4731

Session 3 — Continuation (maybe days later)

You: "Read SPEC.md and continue. Focus on error handling."

Claude: Reads spec (full context restored), continues exactly where we left off.

Summary

INSTEAD OF...	DO THIS...
Explaining requirements in chat	Write them in SPEC.md Section 2
Answering questions in chat	Answer in SPEC.md Section 3
Hoping Claude Code remembers decisions	Record them in SPEC.md Section 4
Re-explaining context each session	Say "Read SPEC.md" at session start

The specification document becomes the project's single source of truth — always available, never forgotten, continuously refined.

---

team@gyreresearch.com ▪ gyreresearch.com ▪ (800) 933-4731