Specification-Driven Workflow for Claude Code

Author: Scott Sykowski (sms@gyreresearch.com) & Claudio

Date/Version: 2025-01-29

A practical guide for managing Claude Code projects using 1) standardized coding guidelines, and 2) a living specification document instead of conversational chat.

This document will focus on the second part: living specification document

---

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 read 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 doesn't necessarily reduce total tokens—you're 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` (or `PROJECT_SPEC.md`, `REQUIREMENTS.md`—whatever fits your conventions) 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

What the system must do:

- FR-1: [Requirement description]
- FR-2: [Requirement description]
- FR-3: [Requirement description]
```


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

2.2 Non-Functional Requirements

Constraints and quality attributes:

- NFR-1: Must run on Rocky Linux 9
- NFR-2: Must complete within 5 minutes
- NFR-3: Must follow Gyre coding standards

2.3 Out of Scope

Explicitly state what this project does NOT include (prevents scope creep):

- *[Thing we're not building]*
- *[Feature deferred to future phase]*

---

3. Q&A Log

Questions from Claude and answers from the owner. Keep chronological with dates.

2025-01-29

Q1 (Claude): How should the system handle API rate limits—retry with backoff, or queue and batch?

A1 (Owner): Retry with exponential backoff, max 3 attempts, then skip and log the failure.

Q2 (Claude): What's the target database—ClickHouse or SQL Server?

A2 (Owner): SQL Server. We migrated off ClickHouse last month.

2025-01-30

Q3 (Claude): *[Next question...]*

A3 (Owner): *[Answer...]*

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

4. Decisions

Architectural and design decisions with rationale. Reference Q&A entries wher
e relevant.

ID	Decision	Rationale	Date
D-1	Use retry with backoff for API calls	Per A1—owner preference for re
silience over speed	2025-01-29
D-2	Target SQL Server	Per A2—infrastructure migration complete	2025-0
1-29
D-3	Single-file script, no separate modules	Project is small; simplicit
y over reusability	2025-01-29

---

5. Current Status

5.1 Completed

- [x] Initial requirements gathered
- [x] Database schema designed
- [x] API integration tested

5.2 In Progress

- [ ] Main ETL script (60% complete)
- [ ] Error handling refinement

5.3 Blocked / Waiting

- [ ] Production credentials (waiting on Scott)

5.4 Next Steps

1. Complete ETL script
2. Add logging per Gyre standards
3. Write self-tests
4. Deploy to cron

---

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

```
## 6. Technical Notes

Any technical details, gotchas, or reference information useful during implem
entation


### API Details

- Endpoint: `https://api.example.com/v2/data`
- Auth: Bearer token in header
- Rate limit: 100 requests/minute

### Database

- Server: `db-prod.internal`
- Database: `analytics`
- Target table: `daily_prices`

### File Locations

- Config: `/etc/gyre/config.json`
- Logs: `/var/log/gyre/`
- Data: `/data/etl/`

---

## Change Log

| Date | Build | Author | Description |
|------|-------|--------|-------------|
| 2025-01-29 | 1 | Your Name | Initial specification |
```

The Workflow

Starting a New Project

1. **Create `SPEC.md`** with your initial requirements (Sections 1-2 minimum)
2. **First Claude Code session:**

   ```
   Read SPEC.md and ask any clarifying questions.
   Append your questions to Section 3 (Q&A Log).
   ```


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

3. **You answer** by editing the Q&A section directly in the file.
4. **Next prompt:**

```
Read the updated SPEC.md and proceed with implementation.
Update Section 5 (Current Status) 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 gives me everything Claude Code need—requirements, prior Q&A, decisions, and current status.

When Questions Arise Mid-Session

If Claude Code need clarification during implementation:

1. I'll append the question to Section 3
2. You answer in the document
3. I'll re-read and continue

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 - Any new constraints or learnings are in Section 6

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.

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

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 ~500 lines in the spec, consider splitting into: - `SPEC-overview.md` - High-level architecture and decisions - `SPEC-module-a.md` - Detailed requirements for module A - `SPEC-module-b.md` - Detailed requirements for module B

Reference across files as needed.

Example Session Flow

Session 1 - Project Kickoff

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

Claude: *Reads spec, appends 5 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*

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*

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

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 remember 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.

Guide Version: 2025-01-29 Build:1

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