# GitHub Spec-Kit — A Practical Guide for Structured AI Development

## 1. Why Spec-Kit Exists

AI coding tools work great when you give them small, isolated tasks. But when you ask them to build something bigger—like a real feature or full application—they often:

- Produce inconsistent output
- Change the tech stack without asking
- Break existing features
- Miss edge cases and UX details
- Lose context when you switch chats

This happens because AI agents **don't naturally follow engineering workflows**.

**SpecKit fixes that by forcing AI to behave like an experienced software engineer**:

- Understand the goal
- Document requirements
- Ask clarifying questions
- Plan architecture
- Break work into tasks
- Implement step-by-step
- Keep everything consistent

It creates structure around the AI to make development predictable, stable, and maintainable.

---

## 2. What Spec-Kit Actually Does

SpecKit gives the AI a **standardized development lifecycle**, so instead of "here's your code," you get:

### ✔ A proper specification

### ✔ A plan and system design

### ✔ Task breakdown

### ✔ Ordered implementation

### ✔ Documentation that stays updated

Instead of AI improvising, SpecKit turns your feature into a **documented, repeatable engineering process**.

---

## 3. The SpecKit Lifecycle

The whole workflow is driven by **slash commands** that your AI agent understands.

```
Constitution → Specify → Clarify → Plan → Tasks → Implement
```

Each step builds on the previous one.

---

## 4. Constitution — Your Project's Operating System

`/speckit.constitution`

This step defines **how the entire project should be built**.

It sets the rules your agent must follow:

- Coding standards
- UX principles
- Architectural preferences
- Testing expectations
- Allowed tech stack
- Project constraints

It’s rarely updated—only when your overall rules change.

This prevents AI from making random tech choices in later steps.

---

## 5. Specification — Define the Feature (Not the Code)

`/speckit.specify`

This is where you tell the agent:

- What the feature should do
- What users expect
- Required behaviors
- Edge cases
- Success criteria

The key point: **No technical decisions yet.** Just functionality and business intent.

SpecKit stores this in a structured spec folder for that feature.

---

## 6. Clarify — Remove Ambiguity

`/speckit.clarify`

The agent reviews your spec and asks targeted questions:

- Missing acceptance criteria
- Edge cases you didn’t consider
- Validation rules
- Error-handling expectations

This ensures the feature is fully understood **before** any planning starts.

---

## 7. Plan — Architecture & Technical Decisions

`/speckit.plan`

Now the AI makes the technical plan, based on:

- Your constitution rules
- The final approved specification

The plan includes:

- System architecture
- Data models
- API contracts
- Chosen libraries & versions
- Folder structure
- Setup instructions
- Step-by-step implementation strategy

This document becomes the technical source of truth.

---

## 8. Tasks — Turning the Plan Into Actionable Units

`/speckit.tasks`

SpecKit breaks the plan into ordered tasks:

- T001, T002, T003…
- With file paths ☒
- Dependency order
- Test-first tasks (if using TDD)
- Parallel tags `[P]` where possible

This gives the AI a professional, developer-style task list.

---

## 9. Implementation — Build Without Guessing

`/speckit.implement`

This is the execution phase:

- AI follows tasks one by one
- Writes tests, then code
- Updates task progress
- Runs commands (npm, migrations, formatting)
- Creates commits logically
- Never jumps ahead or improvises

By the end, you have:

- Working feature
- Complete documentation
- A clean commit history
- A PR-ready branch

---

## 10. Example: Adding a Feature in Next.js

### Step 1 — Initialize your project

```
npx create-next-app .
```

### Step 2 — Add SpecKit

It generates:

- `specify/` directory
- Memory file
- Script hooks
- Spec/plan/task templates

### Step 3 — Constitution

```
/constitution
Follow Next.js 15 best practices, WCAG accessibility, and modular architecture.
```

### Step 4 — Specification

```
/specify
Build an expense tracker with add/view/delete and category filtering.
```

### Step 5 — Clarify

SpecKit asks questions → you answer → spec updates.

### Step 6 — Plan

```
/plan
Use Next.js App Router + local storage + server actions.
```

### Step 7 — Tasks

```
/tasks
```

### Step 8 — Implement

```
/implement
```

The agent implements everything based on the tasks and plan.

---

## 11. Adding Another Feature? Repeat.

Just say:

```
/specify Add budget alerts to the expense tracker.
```

SpecKit:

- Creates a new feature folder
- Generates its own spec → plan → tasks
- Keeps everything isolated in a new branch

No interfering with previous features.

---

## 12. When Should You Use SpecKit?

### Great For:

- Production features
- Client deliverables
- Complex flows
- Multiple developers
- Anything requiring documentation
- Stable long-term projects

### Not Needed For:

- Throwaway prototypes
- Small scripts
- Learning projects
- Quick experiments

---

## 13. Final Summary

SpecKit takes what normally happens inside a software engineering team and **turns it into an AI-powered workflow**.

It gives you:

- Structure
- Documentation
- Predictability
- Safer iteration
- Cleaner code
- Traceable decisions
- Production-quality output

It stops the chaos of “AI coding” and replaces it with a **real development lifecycle**, guided by your rules and intentions.