Red Hat

Agor + Speckit + Jira

Multi-Agent Specification-Driven Development

AI-Assisted Development with Structure and Scale

An attempt at an Agentic Engineering Workflow

TL;DR

Humans Are the Bottleneck

The Challenge: We—humans—are the bottleneck in the development process.

The Opportunity: Let's find systematic ways to address this challenge using AI-assisted workflows.

The Problem

Traditional AI-Assisted Development Challenges

🎯 Ad-hoc Prompting

❓ Ambiguous Requirements → Rework Cycles

🔀 Lost Context & Misaligned Implementation

😫 Review Fatigue ***

The "What, Not How" Principle

Critical Success Factor for AI-Assisted Development

Clarity about WHAT to build is the highest-leverage work in the entire workflow.

For the HOW - build guardrails and constraints with common practices.

Outsource the *writing* not the *thinking*.

Vague vs. Clear Requirements

❌ Vague (Causes Problems)	✅ Clear (Enables Succ
"Add user authentication"	"Users must log in with email/password. Session expires after 24 hours. Failed attempts are rate-limited to 5 min
"Make it faster"	"Page load time must be under 2 seconds for 95th percentile users on 3G connecti
"Handle errors better"	"All API errors return structured JSON with error code, message, and correlation ID. 4xx errors are logged at WARN, at ERR
"Add tests"	"Unit test coverage must reach 80%. All public APIs require integration tests. Edge cases from acceptance criteria r have explicit test ca

made with marimo

Front-Load the Thinking

The Speckit workflow is designed to front-load clarity:

1. **Specify** captures WHAT the feature does and WHY it matters
2. **Clarify** surfaces ambiguities BEFORE technical decisions
3. **Plan** translates clear requirements into HOW to build
4. **Tasks** break the plan into executable units

Investing in Specify and Clarify stages isn't optional overhead—it's the highest-leverage work.

The Solution: Three Tools

Agor / Speckit / Jira

🎨 Agor - Multi-Agent Canvas for Orchestration

Key Capabilities:

- 📊 Visual workflow management and session coordination
- 🔀 Fork/spawn patterns for parallel AI work
- 👥 Team collaboration with shared worktrees
- 💾 Checkpoint and restore session capabilities

Learn More: agor.live

made with marimo

📋 Speckit - Structured Workflow Framework

Key Capabilities:

- ⚙️ 7-stage specification-driven development process
- ⭐ 62,000+ GitHub stars - battle-tested workflow
- 📝 Systematic transformation from requirements to code
- 🎯 Constitution-based team standards enforcement

Learn More: github.com/github/spec-kit

🎫 Jira - Issue Tracking Integration

Key Capabilities:

- 🏢 Common platform for company requirements & prioritization
- 🔗 Automatic ticket extraction from branch names
- 📈 Draft PR creation and bidirectional linking
- 📋 Context fetching (parent stories, subtasks, full history)

Status: Company Standard Tool

How They Work Together

```
Jira Ticket → Agor Canvas → Speckit Workflow → Claude AI → Git Repository
    ↑                                                          ↓
    └─────────────────── PR Updates ───────────────────────────┘
```

made with marimo

The workflow: Jira provides requirements → Agor orchestrates multiple AI sessions → Speckit enforces structured stages → Cha
tracked in Git → Progress visible in Jira

Human-in-the-Loop Review Gates

AI Assists, Humans Decide - Speckit Workflow

```
┌───────┐    ┌───────┐    ┌───────┐    ┌────┐    ┌─────┐    ┌───────┐    ┌─────────
│ Setup │───▶│Specify│───▶│Clarify│───▶│Plan│───▶│Tasks│───▶│Analyze│───▶│Implement
└───────┘    └───────┘    └───────┘    └────┘    └─────┘    └───────┘    └─────────
                 │            │           │         │           │           │ │
                 ▼            ▼           ▼         ▼           ▼           ▼
             [REVIEW]     [REVIEW]    [REVIEW]  [REVIEW]    [REVIEW]    [REVIEW]
```

What to Review at Each Stage

Stage	Review Focus	Key Questions
Specify	Requirements accuracy	Does this capture the real need? Would stakeholders agree?
Clarify	Completeness	Are all ambiguities resolved? Any remaining questions?
Plan	Technical soundness	Is this the right approach? What are the risks?
Tasks	Scope & granularity	Are tasks properly sized? Dependencies clear?
Analyze	Consistency	Do spec, plan, and tasks align? Any gaps?
Implement	Quality	Does code match plan? Tests passing?

Feedback Loop Options

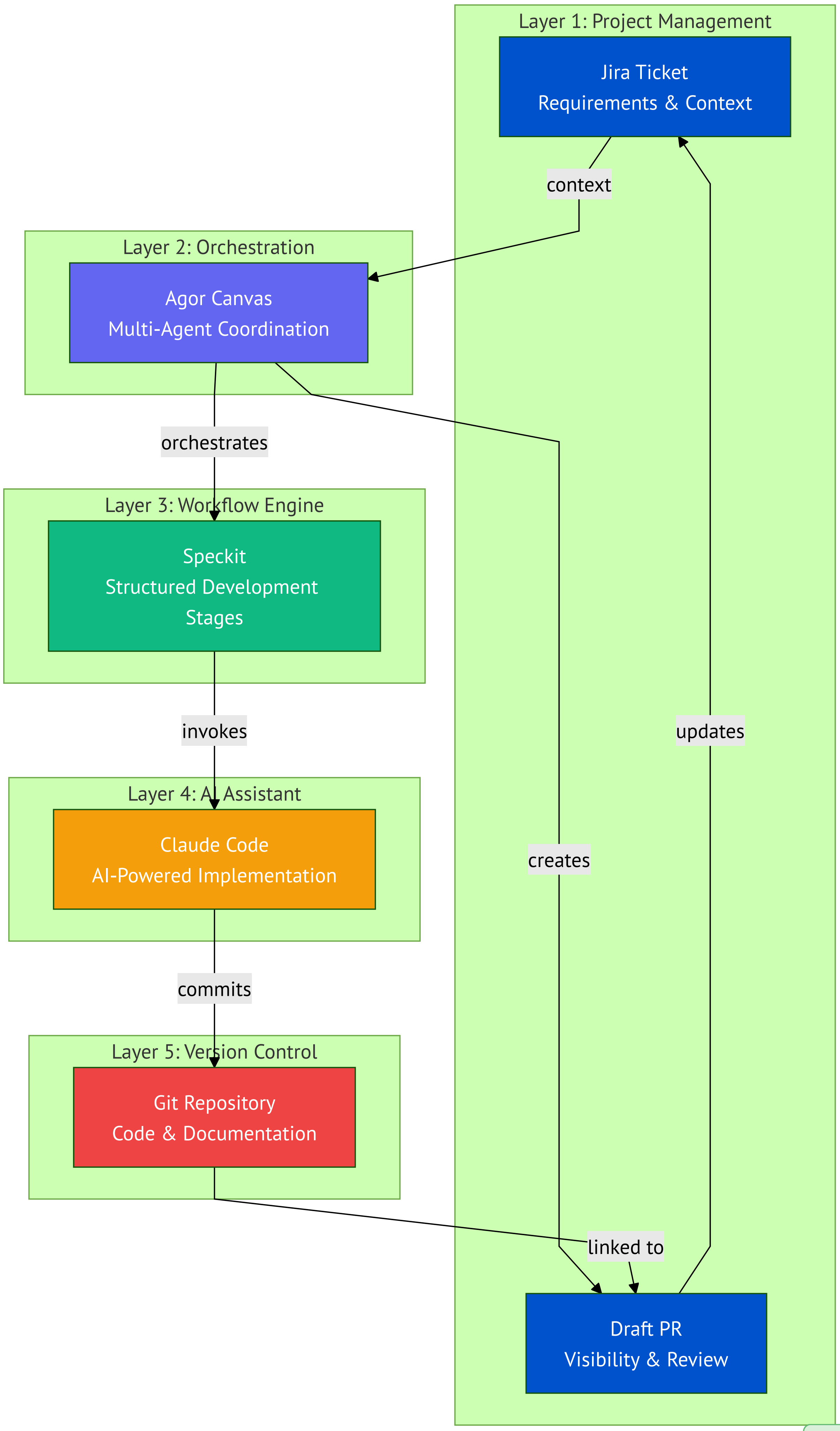
After review, you can:

- ✅ **Approve**: Proceed to the next stage
- 📝 **Revise**: Ask AI to update with your feedback
- 🔄 **Iterate**: Go back to a previous stage
- 🔀 **Fork**: Explore alternatives while preserving current approach

All stages are easily referenced and visible in the Agor canvas.

Architecture Overview

Data Flow Between Layers


Layer 1: Project Management
Jira Ticket
Requirements & Context
context
Layer 2: Orchestration
Agor Canvas
Multi-Agent Coordination
orchestrates
Layer 3: Workflow Engine
Speckit
Structured Development
Stages
invokes
Layer 4: AI Assistant
Claude Code
AI-Powered Implementation
commits
Layer 5: Version Control
Git Repository
Code & Documentation
creates
updates
linked to
Draft PR
Visibility & Review

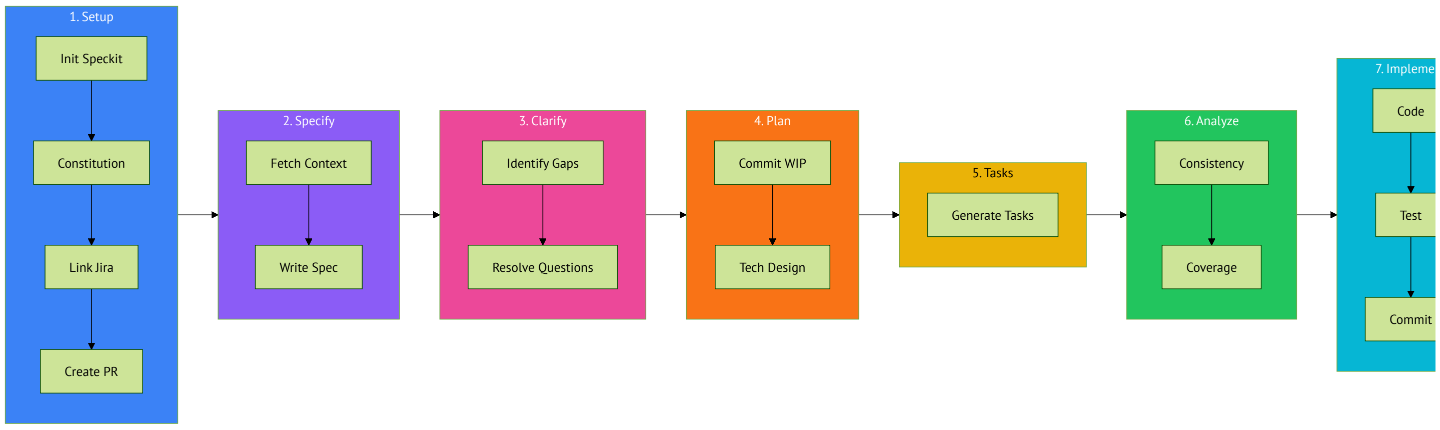
Layer Responsibilities

1. **Project Management**: Jira tracks requirements, PRs provide visibility
2. **Orchestration**: Agor coordinates multiple AI sessions visually
3. **Workflow Engine**: Speckit enforces structured development stages
4. **AI Assistant**: Claude Code executes implementation with AI
5. **Version Control**: Git stores all artifacts and code

Data flows bidirectionally, ensuring all layers stay synchronized.

The 7-Stage Workflow

Visual Overview



Stages at a Glance

1. **Setup** → repo/branch/Constitution
2. **Specify** → Requirements/sync with Jira
3. **Clarify** → Resolve Ambiguities
4. **Plan** → Technical Design
5. **Tasks** → Work Breakdown
6. **Analyze** → Consistency Check
7. **Implement** → Execution

Best Practices

Setting Up for Success

Critical Foundation: Two Key Artifacts

1. Team Constitution Document

A comprehensive `constitution.md` that captures:

- Company coding standards and conventions
- Team-specific architectural patterns
- Technology stack decisions and rationale
- Testing requirements and coverage expectations
- Documentation standards
- Code review guidelines
- Security and compliance requirements

2. Well-Defined Jira Tickets

Each ticket should contain:

- Clear problem statement or feature description
- **Specific, testable acceptance criteria**
- Definition of Done (DoD) aligned with team standards
- Relevant context, background, and business justification
- Links to related tickets, documentation, or designs

The quality of these artifacts directly impacts the quality of AI-generated specs, plans, and code.

made with marimo

🎯 Catch Vague Requirements Up Front

👥 Review Specifications with Stakeholders

🔄 Commit Before Stage Transitions

📋 Use Draft PRs Early

Challenges & Path to Success

What We've Learned

✅ Easy Wins - Start Here

1. **Common AGENT.md with well-defined rules**
   - Single source of truth for AI behavior
   - Consistent across all team repositories
   - Versioned and reviewed like code
2. **Well-thought-out constitution.md for team and repo**
   - Captures institutional knowledge
   - Reduces "how do we do X?" questions
   - Improves AI-generated code quality immediately
3. **Technical design should have more guidance from repo standards**
   - Include architecture decision records (ADRs)
   - Document common patterns and anti-patterns
   - Provide examples of good implementations

⚠️ Challenge: Initial Setup Time

⚠️ Challenge: Team Adoption

⚠️ Challenge: Keeping Artifacts in Sync

⚠️ Challenge: Balancing Automation vs Control

Key Takeaways

What to Remember

🎯 Transforms Ad-Hoc AI Prompting into Structured Process

Move from "hope the AI understands" to systematic, repeatable workflow with clear stages and checkpoints.

📊 Front-Load Clarity to Reduce Rework

Investing in Specify and Clarify stages prevents 3-5x rework downstream. The highest-leverage work is defining WHAT to build, not HOW to build it.

👥 Human Review Gates at Every Stage

AI assists, humans decide. Each stage transition is a deliberate checkpoint where you review, validate, and approve before proceeding. All stages are visible and easily referenced.

made with marimo

🔄 Multi-Agent Orchestration Scales Work

Agor's fork/spawn patterns enable parallel execution while maintaining context and consistency. Visual canvas makes complex workflows manageable.

📝 Context Preservation Throughout Workflow

Jira integration, draft PRs, and staged artifacts create single source of truth. No more lost context or misaligned implementations.

Resources & Next Steps

Get Started Today

🔗 Tool Links

- **Agor**: agor.live
  - Multi-agent orchestration canvas
  - Advanced Features Guide
- **Speckit**: github.com/github/spec-kit
  - 62K+ stars
  - Specification-driven development framework
- **Claude Code**: Anthropic Documentation
  - AI-powered development assistant

📚 Documentation

- **Workflow Guide**: `guides/agor-speckit-workflow.md`
- **Templates**: `guides/agor-speckit-templates.md`
- **Diagrams**: `guides/agor-speckit-workflow-diagram.md`

🚀 Getting Started

1. **Create your constitution.md** - Document team standards
2. **Set up AGENT.md** - Define AI behavior rules
3. **Pick a Jira ticket** - Choose well-defined requirement
4. **Run through Setup stage** - Initialize Speckit + Agor
5. **Follow the 7 stages** - Trust the process

Start small, iterate, and scale what works.

Red Hat

An Attempt at an Engineering Workflow Automation

made with marimo