AI-Driven SDLC: Claude Code Team Collaboration Guide

> **Executive Summary for Stakeholders** | Complete workflow guide for the entire software development lifecycle using Claude Code as an AI agent collaborator.

Version: 1.0 | **Date:** 2026-01-20 | **Classification:** Internal - All Staff

Executive Summary (TL;DR)

BravoSUITE has implemented an **AI-first software development process** where Claude Code acts as an intelligent team member that collaborates with every role across the SDLC. This document explains how each team member (PO, BA, Developer, QA, QC, PM) can leverage Claude Code to accelerate delivery while maintaining quality standards.

Key Value Proposition (Estimated)

Metric	Traditional	With Claude AI	Improvement
Idea → PBI	2-3 days	2-4 hours	~80-90% faster
PBI → Test Cases	1-2 days	1-2 hours	~85% faster
Bug Investigation	4-8 hours	30-60 min	~70-85% faster
Documentation	2-4 hours	15-30 min	~85% faster
Code Review	1-2 hours	15-30 min	~75% faster

Estimates based on initial pilot usage. Actual results vary by task complexity.

What Makes This Different

1. **Single Workspace, All Roles** - Every team member uses the same Claude Code environment
2. **Automatic Workflow Detection** - Claude detects your intent and suggests the right workflow
3. **Artifact Continuity** - Ideas flow seamlessly from PO → BA → Dev → QA → Release

4. **Multilingual Support** - Commands work in English, Vietnamese, Chinese, Japanese, Korean
5. **Self-Improving System** - Claude learns your team's patterns and preferences

Why This Matters (Strategic Value)

For Leadership: This isn't just a tool—it's a **force multiplier** for your entire engineering organization.

Strategic Benefit	Business Impact
Reduced Time-to-Market	Features ship faster without sacrificing quality
Knowledge Democratization	Junior team members work at senior velocity
Process Consistency	Every role follows the same workflow, every time
Institutional Memory	Claude learns your patterns, preserving team knowledge
Scalability	Add team members without proportional onboarding cost

The Transformation:

```
Before: Ideas wait in queues → Manual handoffs → Knowledge silos → Inconsistent quality
After:  Ideas flow instantly → Automated handoffs → Shared context → Standardized output
```

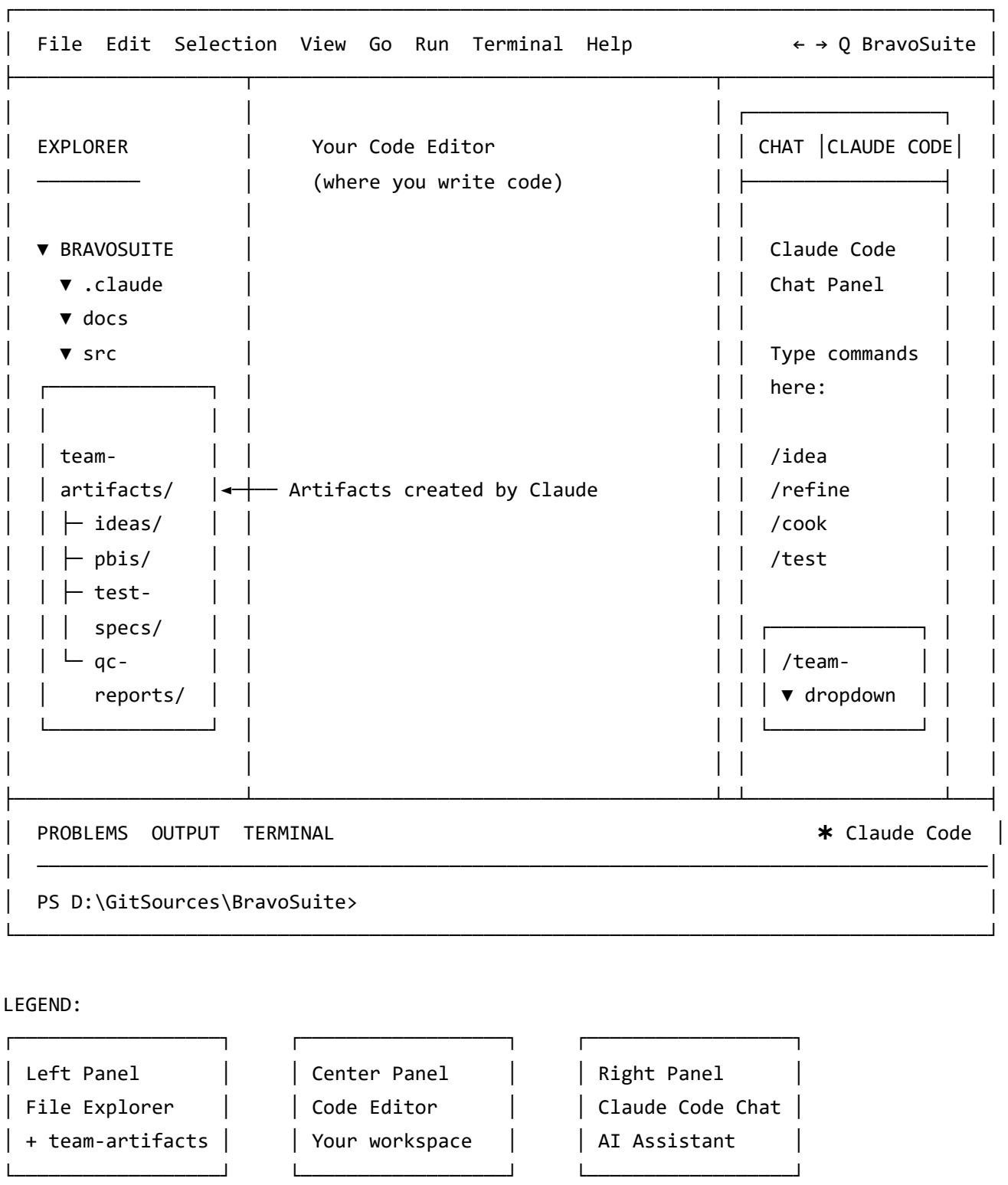
Table of Contents

#	Section	Audience	Time
7	Example Scenarios	All	5 min
8	Complete Workflow Reference	Technical	5 min
9	Adoption Roadmap	Leadership	3 min
10	Command Quick Reference	Daily Use	Reference
11	Setup & Prerequisites	Technical	5 min
12	Troubleshooting	Technical	Reference
13	Appendix: Technical Architecture	Technical	10 min

Getting Started: VS Code Visual Guide

This section provides a visual walkthrough of how to use Claude Code in VS Code.

VS Code Interface Overview



LEGEND:

How to Open Claude Code Chat

Method 1: Keyboard Shortcut

```
Press:  Ctrl + Alt + I  (Windows/Linux)
        Cmd + Alt + I   (Mac)
```

Method 2: Command Palette

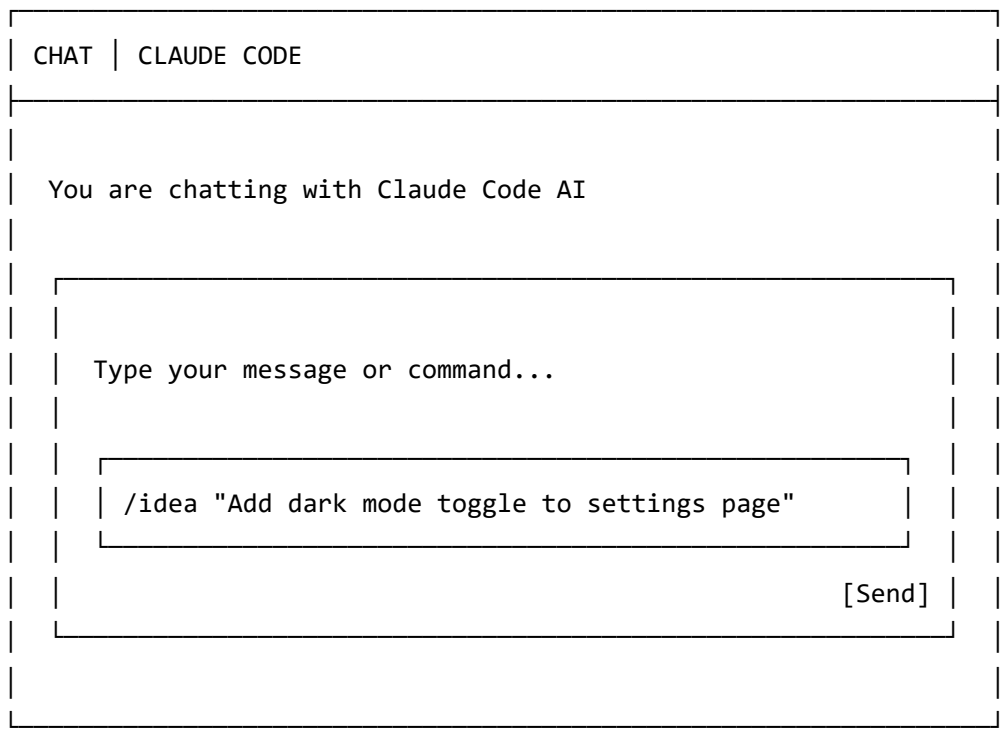
```
Press:  Ctrl + Shift + P  →  Type: "Claude Code"  →  Select: "Open Chat"
```

Method 3: Status Bar

```
Click on "✱ Claude Code" in the bottom-right status bar
```

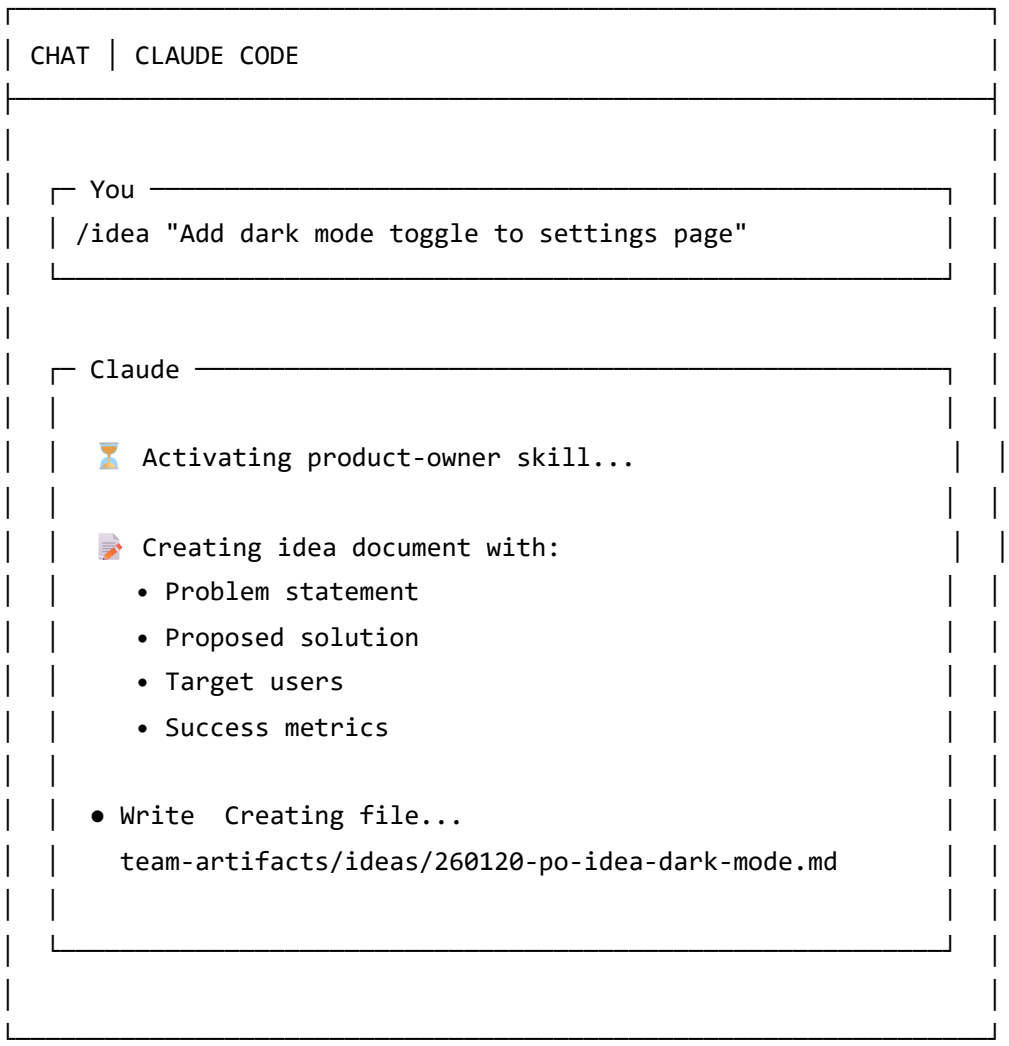
Visual Example 1: Product Owner Creates an Idea

Step 1: Open Claude Code and Type Your Command

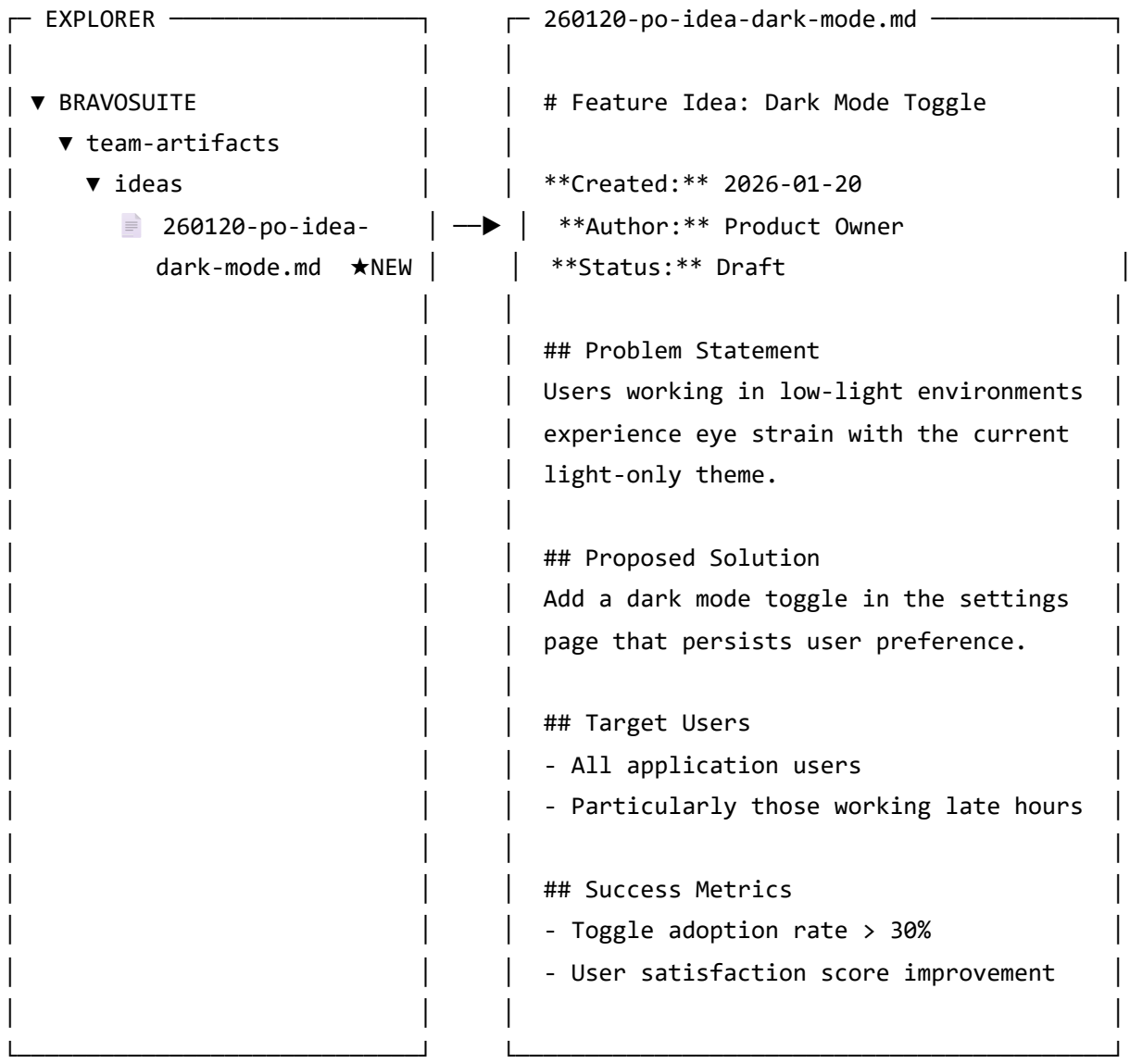



```
YOUR INPUT:
┌──────────────────────────────────────────────────────────────────┐
│  /idea "Add dark mode toggle to settings page"                   │
└──────────────────────────────────────────────────────────────────┘
```

Step 2: Claude Processes and Shows Progress



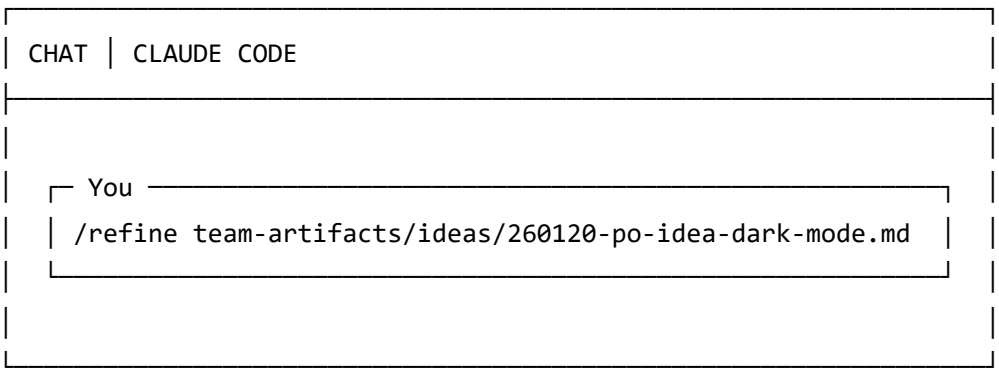
Step 3: Claude Creates the Artifact File



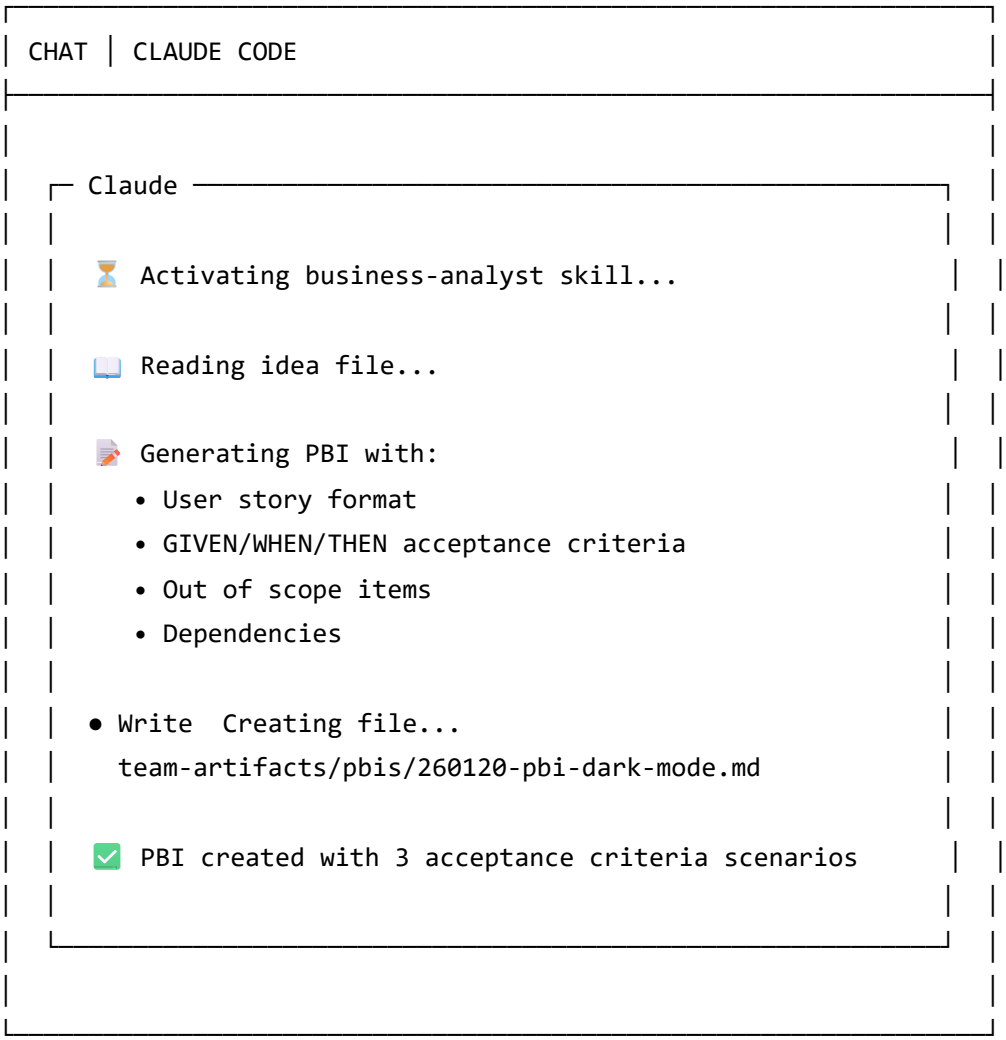
RESULT: New file created in team-artifacts/ideas/

Visual Example 2: Business Analyst Refines Idea to PBI

Step 1: BA Uses /refine Command with the Idea File



Step 2: Claude Generates PBI with Acceptance Criteria

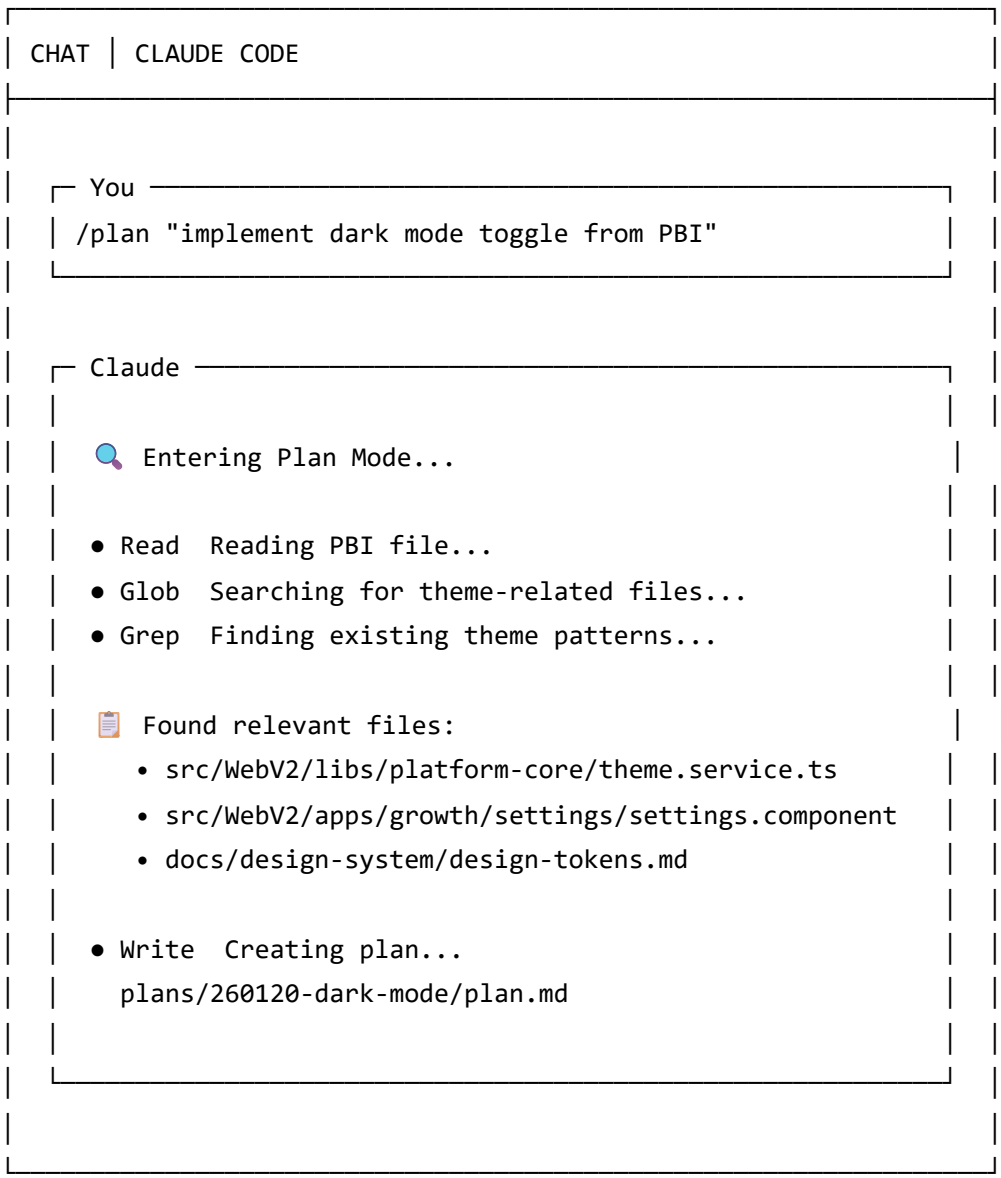


Step 3: Generated PBI Content

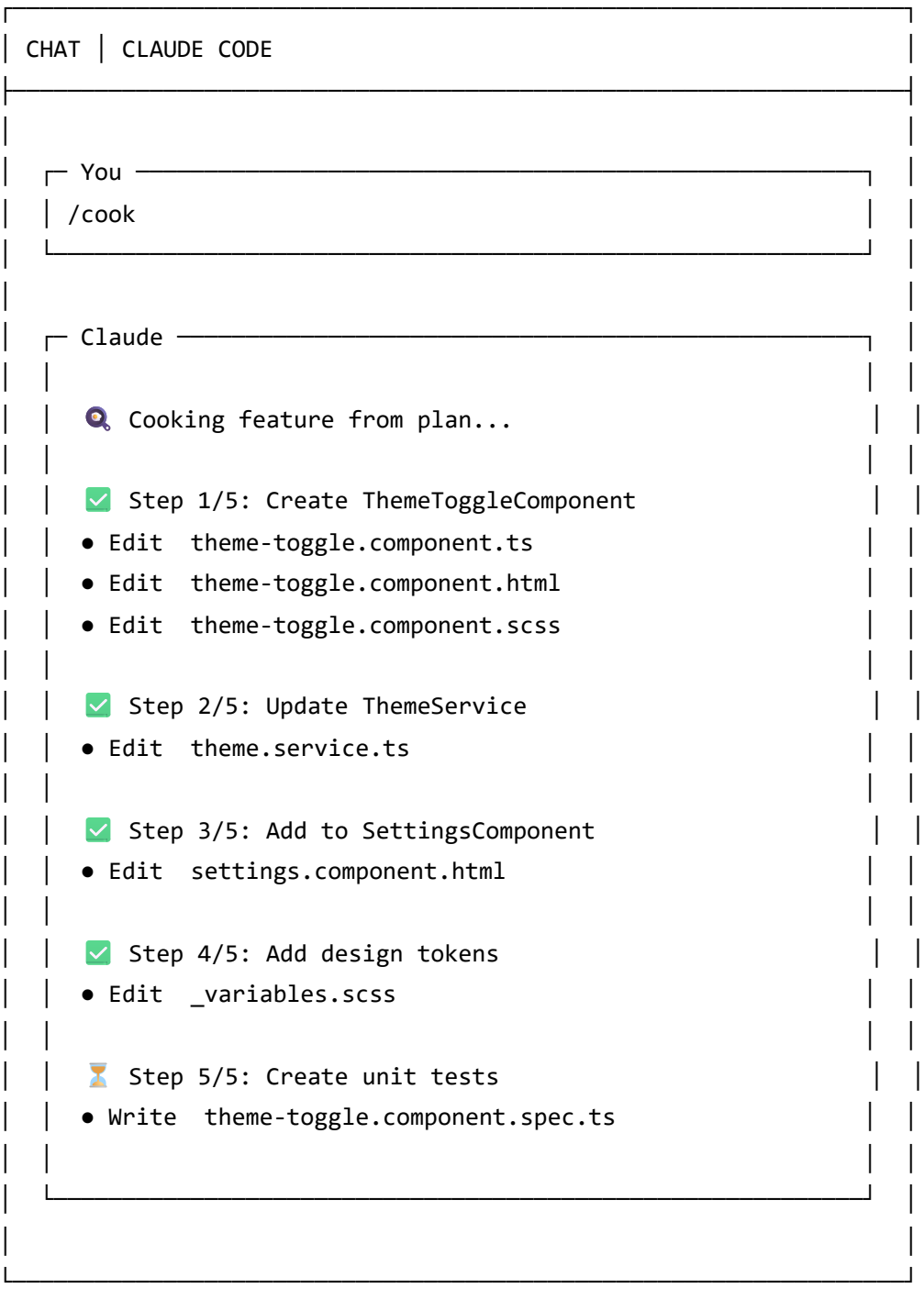
┌─ 260120-pbi-dark-mode.md ──┐
│   │
│ # PBI: Dark Mode Toggle for Settings Page   │
│   │
│ **Source Idea:** 260120-po-idea-dark-mode.md   │
│ **Created:** 2026-01-20 | **Status:** Ready for Development   │
│   │
│ ## User Story   │
│ As a user, I want to toggle between light and dark mode in settings   │
│ so that I can reduce eye strain when working in low-light environments.│
│   │
│ ## Acceptance Criteria   │
│   │
│ ### Scenario 1: Toggle activates dark mode   │
│ ```gherkin   │
│ GIVEN user is on the settings page   │
│ AND current theme is "light"   │
│ WHEN user clicks the dark mode toggle   │
│ THEN theme changes to "dark" within 200ms   │
│ AND all UI components reflect the dark theme   │
│ ```   │
│   │
│ ### Scenario 2: Preference persists across sessions   │
│ ```gherkin   │
│ GIVEN user has set theme to "dark"   │
│ WHEN user logs out and logs back in   │
│ THEN theme should still be "dark"   │
│ ```   │
│   │
│ ### Scenario 3: System respects OS preference   │
│ ```gherkin   │
│ GIVEN user has not set a theme preference   │
│ AND user's OS is set to dark mode   │
│ WHEN user opens the application   │
│ THEN application should default to dark theme   │
│ ```   │
│   │
│ ## Out of Scope   │
│ - Custom color themes   │
│ - Scheduled theme switching   │
│   │
│ ## Dependencies   │
│ - Design system tokens for dark theme   │

Visual Example 3: Developer Implements Feature

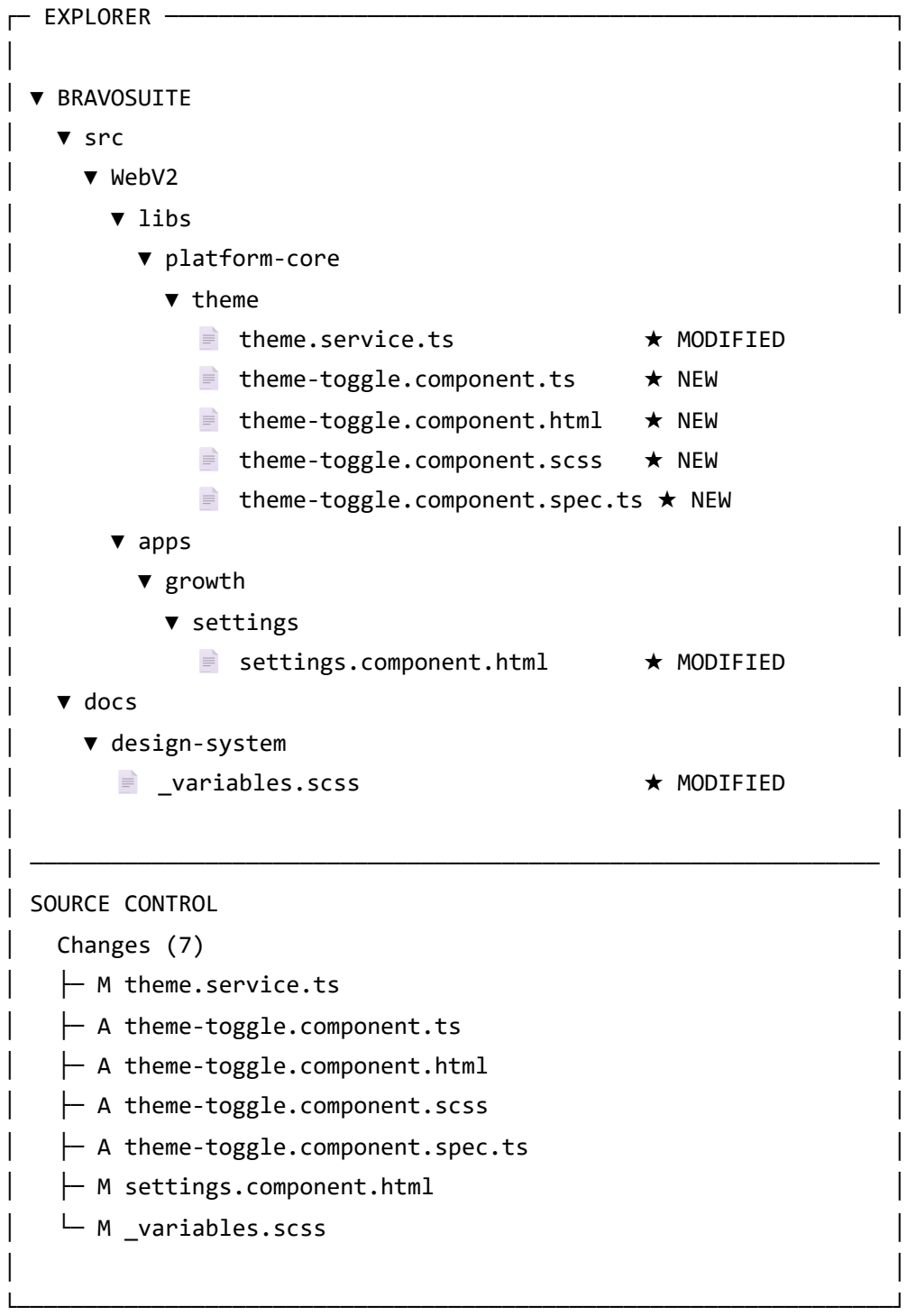
Step 1: Developer Plans the Implementation



Step 2: Developer Executes the Plan with /cook



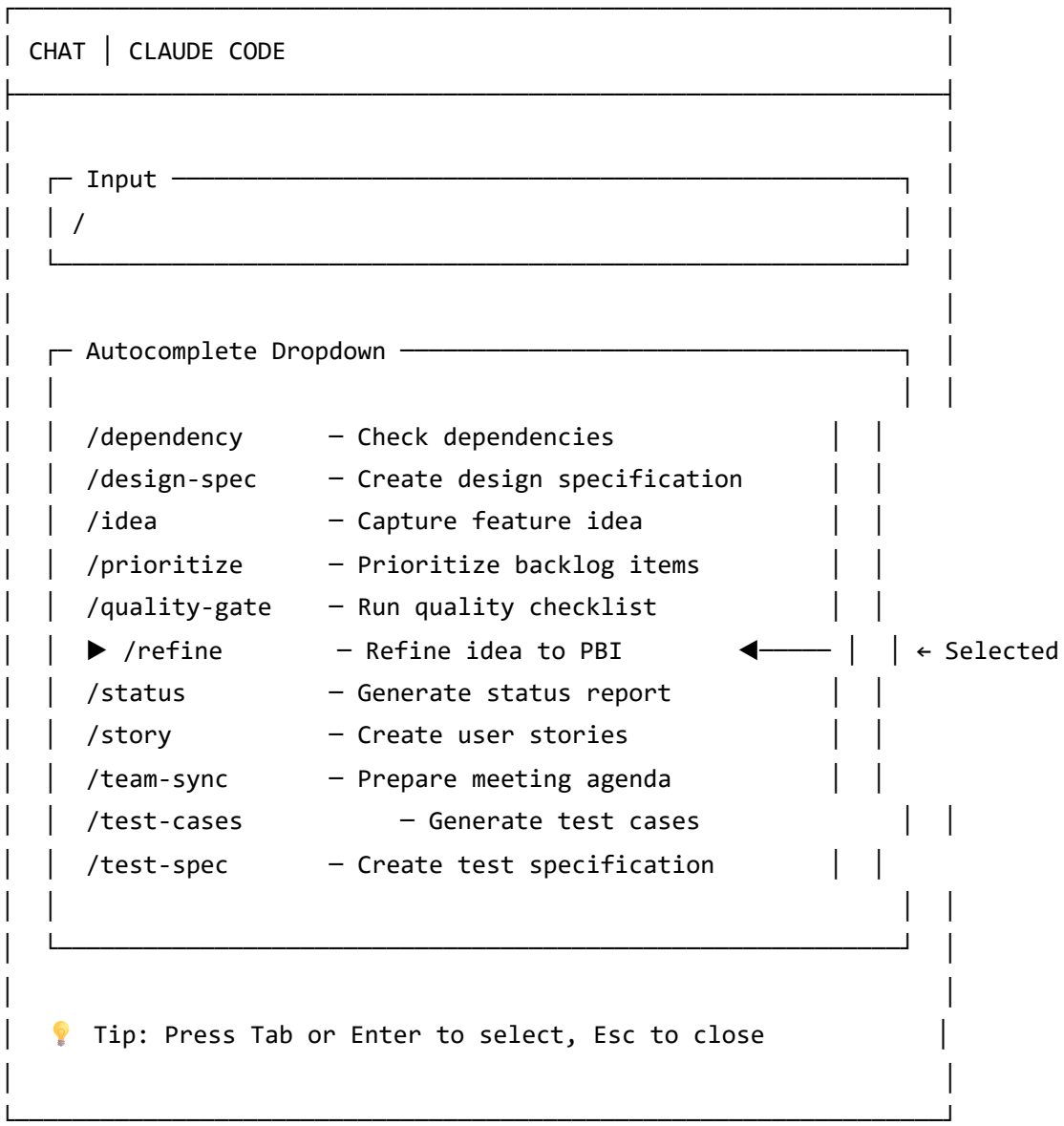
Step 3: Files Created/Modified (Visible in Explorer)



LEGEND: M = Modified, A = Added (New)

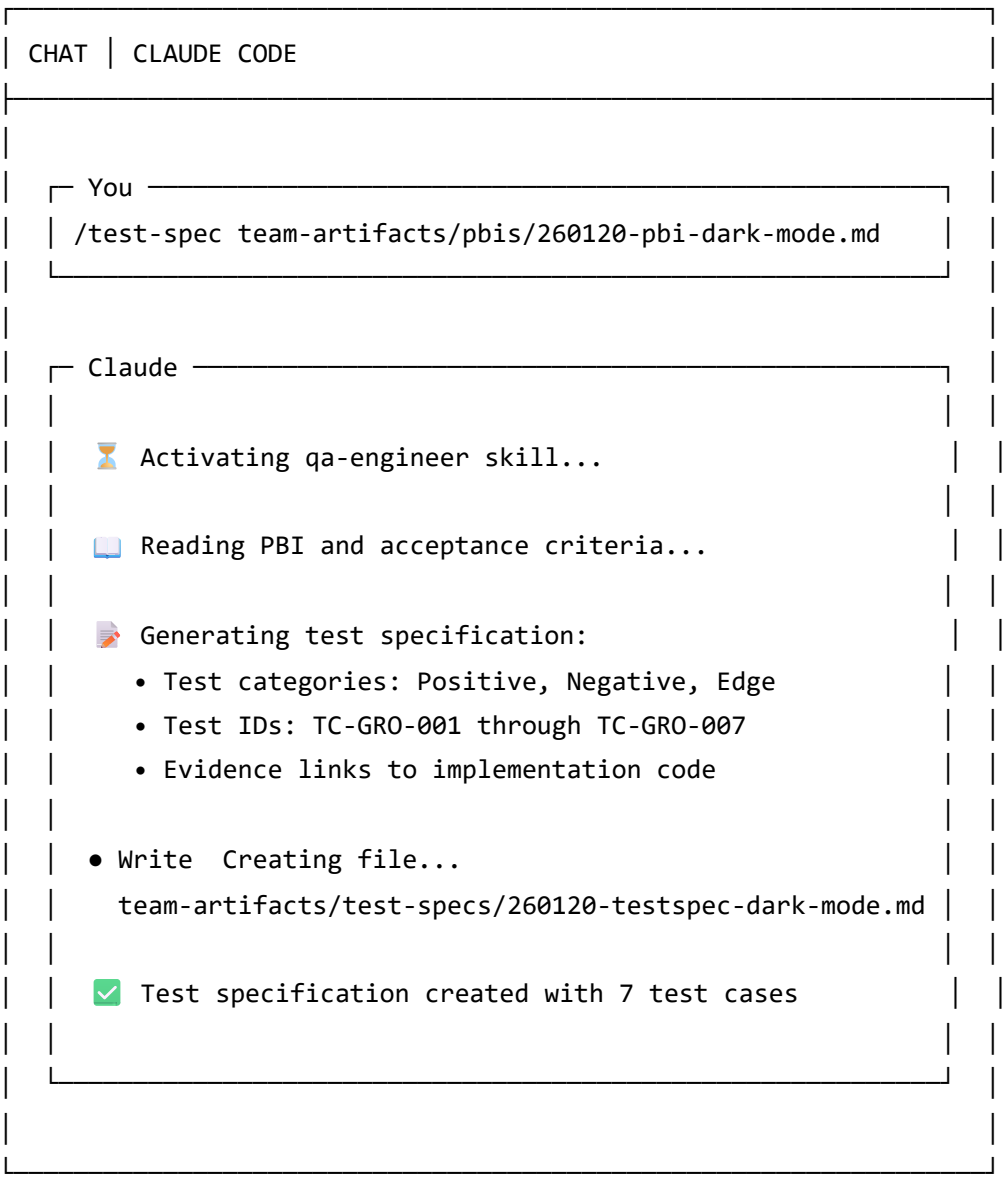
Visual Example 4: Using Command Autocomplete

Type `/team-` to See Available Team Commands



Visual Example 5: QA Creates Test Specification

Step 1: QA Uses /test-spec with PBI

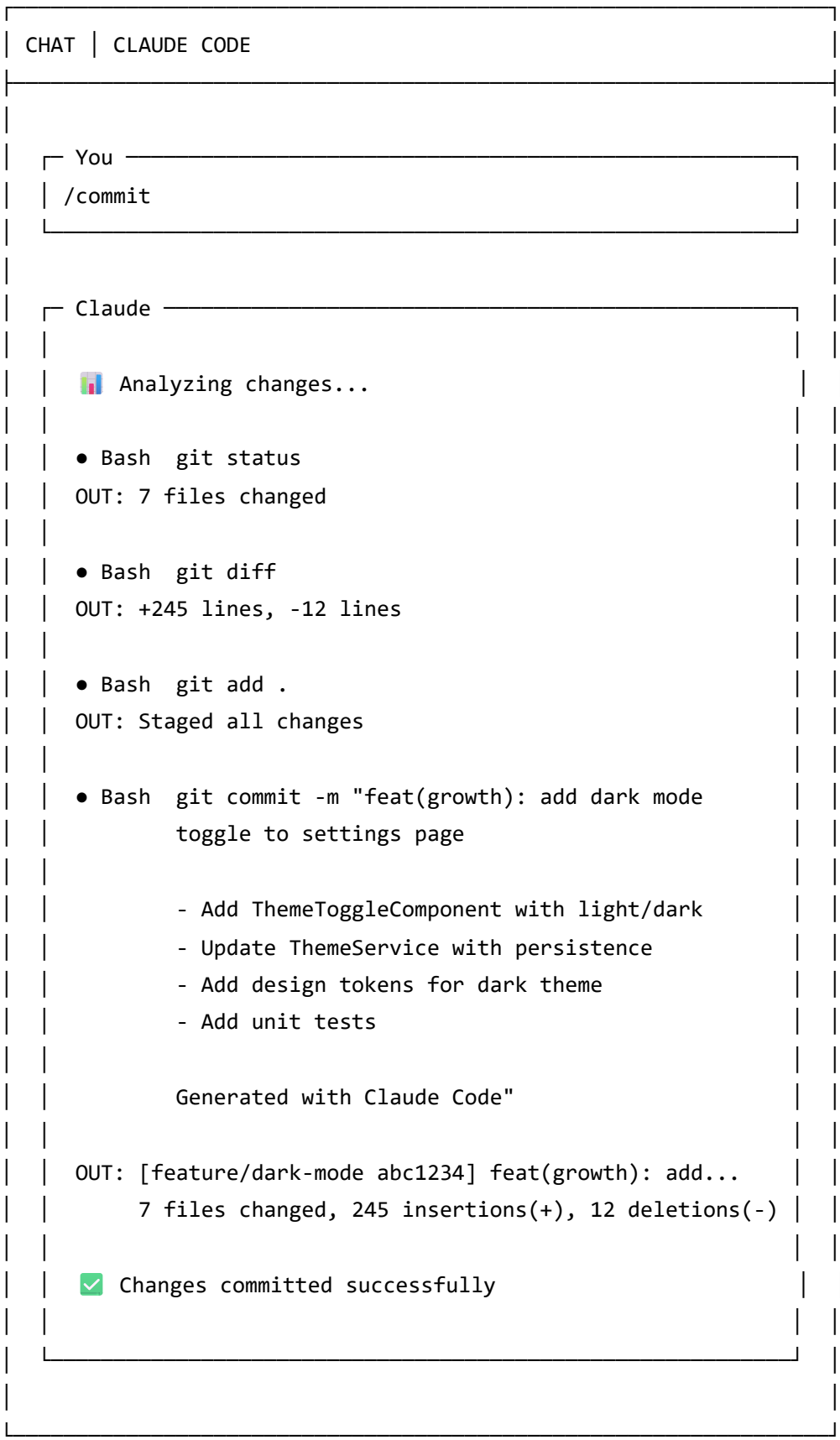


Step 2: Generated Test Specification

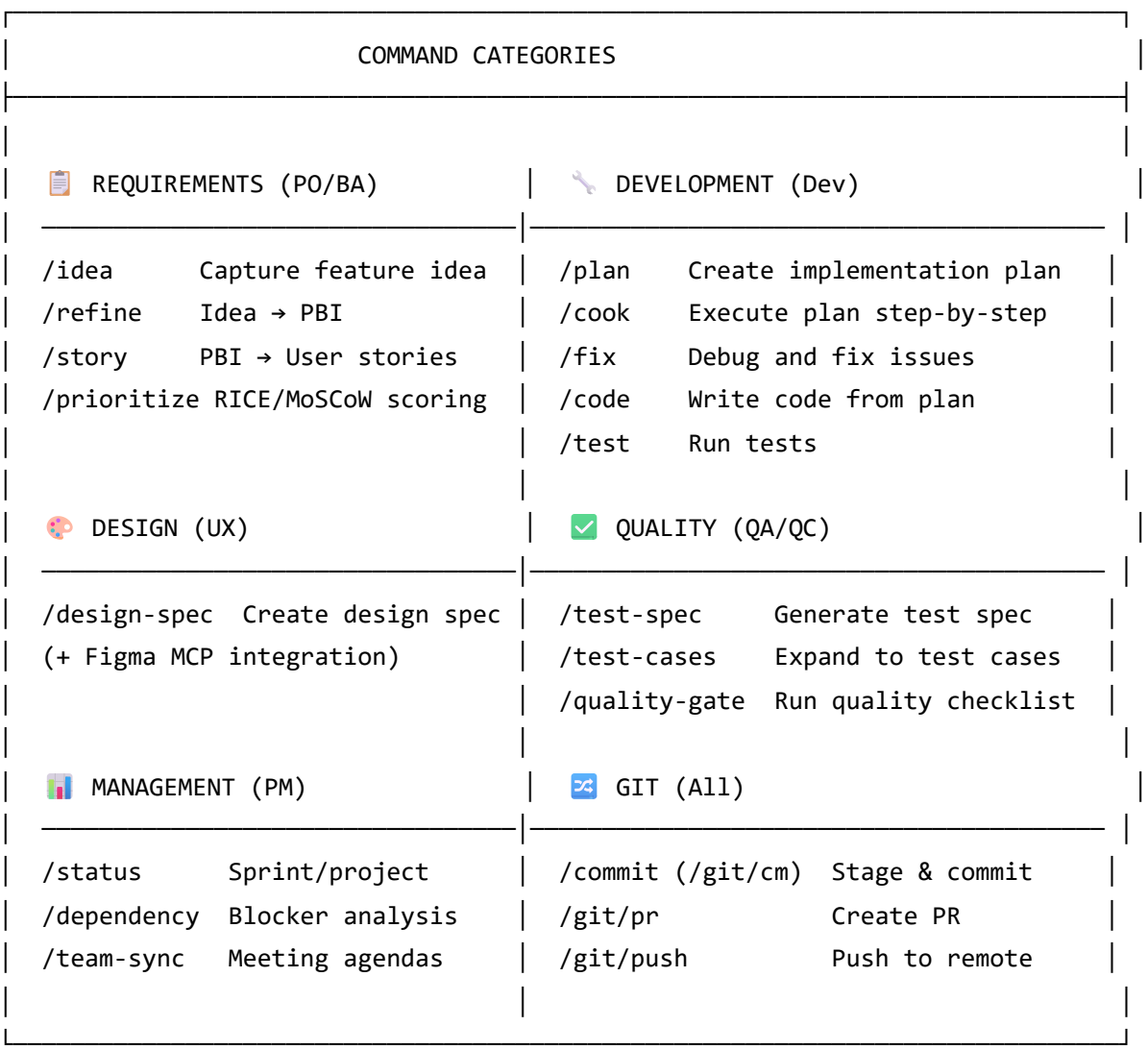
```
┌─ 260120-testspec-dark-mode.md ──────────────────────────────────────────────┐
│                                                                             │
│  # Test Specification: Dark Mode Toggle                                     │
│                                                                             │
│  **Source PBI:** 260120-pbi-dark-mode.md                                    │
│  **Created:** 2026-01-20 | **QA Engineer:** [Name]                          │
│                                                                             │
│  ## Test Cases                                                              │
│                                                                             │
│  ### TC-GRO-001: Toggle activates dark mode (Positive)                      │
│  | Field | Value |                                                          │
│  |-------|-------|                                                          │
│  | Priority | High |                                                       │
│  | Category | Positive |                                                   │
│  | Precondition | User on settings page, theme is light |                   │
│  | Steps | 1. Click toggle |                                               │
│  | Expected | Theme changes to dark within 200ms |                         │
│  | Evidence | `theme-toggle.component.ts:42` |                             │
│                                                                             │
│  ### TC-GRO-002: Toggle activates light mode (Positive)                     │
│  ...                                                                        │
│                                                                             │
│  ### TC-GRO-003: Preference persists after logout (Positive)               │
│  ...                                                                        │
│                                                                             │
│  ### TC-GRO-004: Invalid theme value handled (Negative)                     │
│  ...                                                                        │
│                                                                             │
│  ### TC-GRO-005: OS preference respected when no user pref (Edge)           │
│  ...                                                                        │
│                                                                             │
└─────────────────────────────────────────────────────────────────────────────┘
```

Visual Example 6: Git Integration - Committing Changes

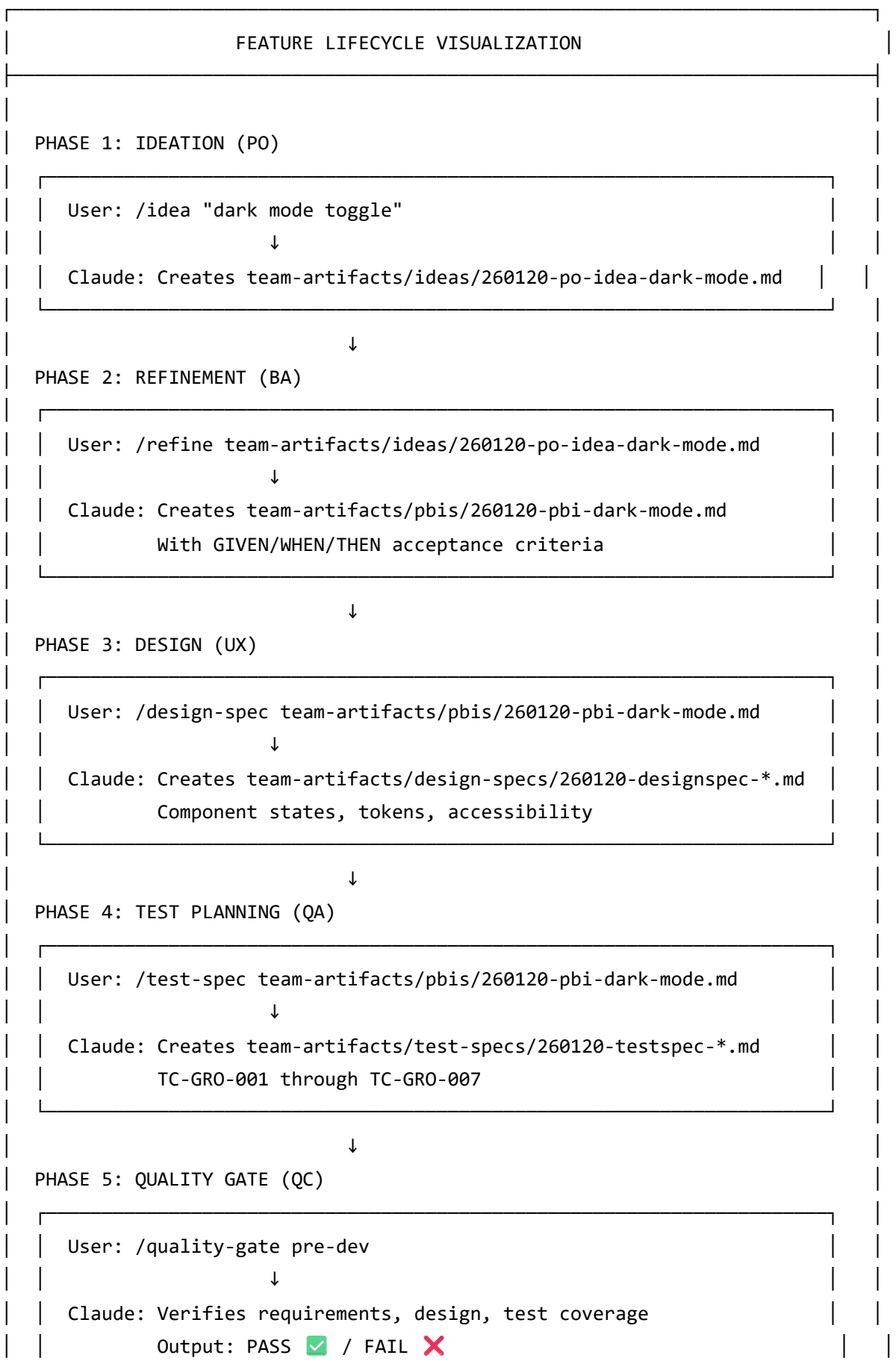
Using /commit (or /git/cm) to Stage and Commit

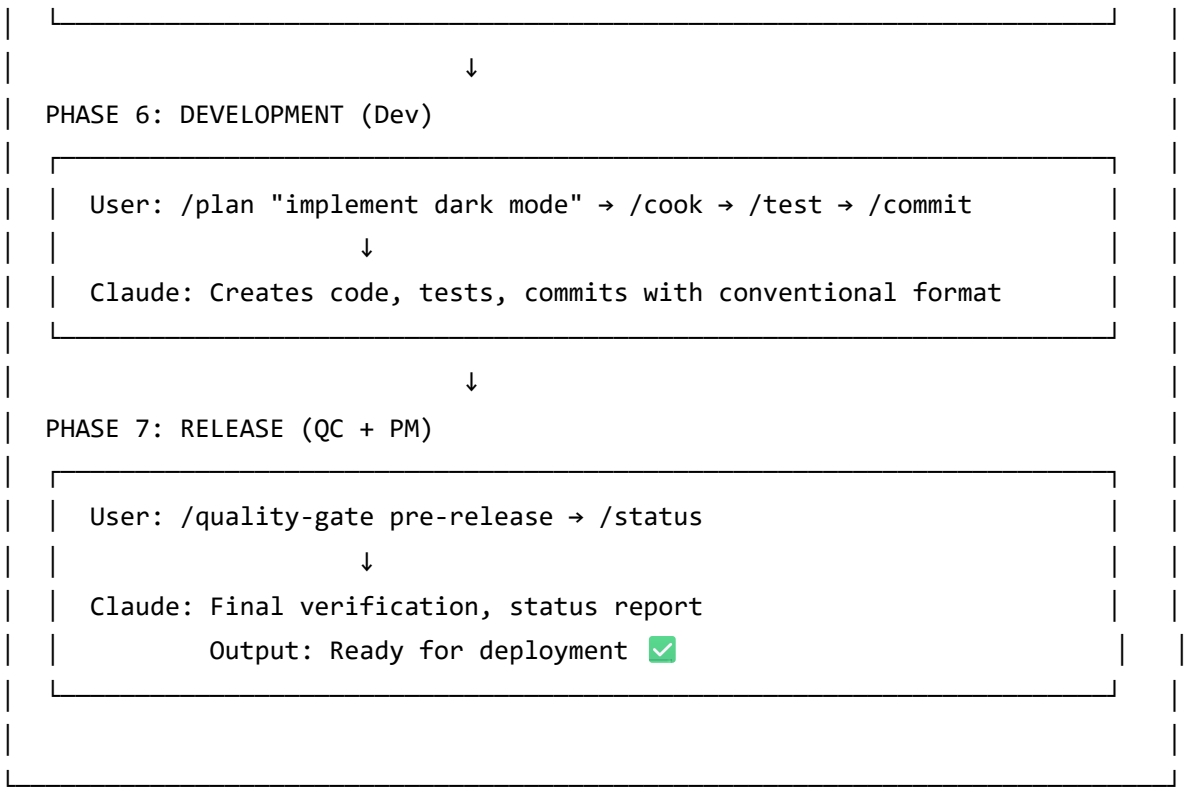

CHAT | CLAUDE CODE
You
/commit
Claude
Analyzing changes...
● Bash git status
OUT: 7 files changed
● Bash git diff
OUT: +245 lines, -12 lines
● Bash git add .
OUT: Staged all changes
● Bash git commit -m "feat(growth): add dark mode
toggle to settings page
- Add ThemeToggleComponent with light/dark
- Update ThemeService with persistence
- Add design tokens for dark theme
- Add unit tests
Generated with Claude Code"
OUT: [feature/dark-mode abc1234] feat(growth): add...
7 files changed, 245 insertions(+), 12 deletions(-)
Changes committed successfully

Command Categories Quick Visual Reference



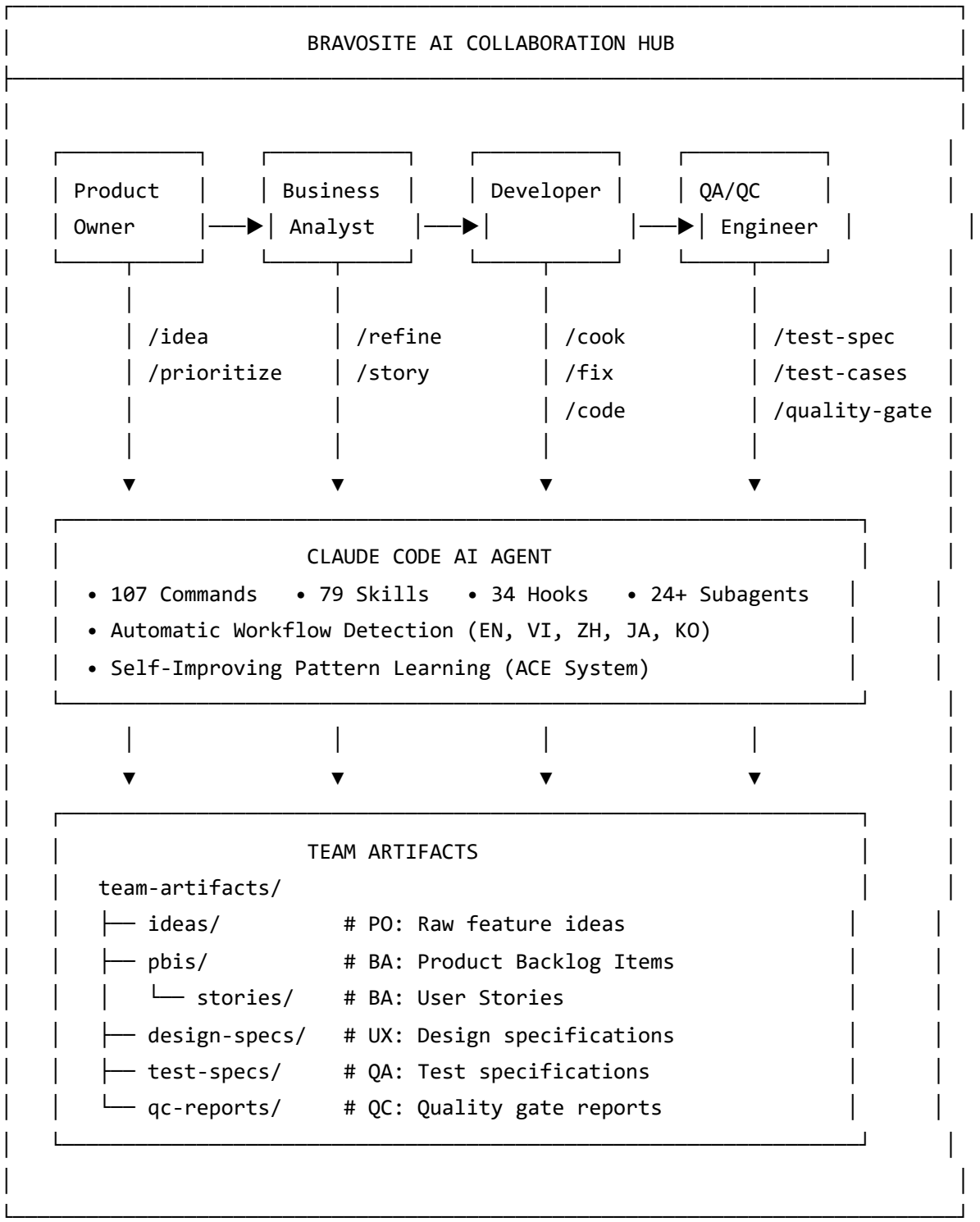
Workflow Visualization: Full Feature Lifecycle




PHASE 6: DEVELOPMENT (Dev)
User: /plan "implement dark mode" → /cook → /test → /commit
Claude: Creates code, tests, commits with conventional format
PHASE 7: RELEASE (QC + PM)
User: /quality-gate pre-release → /status
Claude: Final verification, status report
Output: Ready for deployment ✅

System Overview

How Claude Code Works in BravoSUITE



Core Components

Component	Count	Purpose
Commands	107	Explicit actions invoked with `/` prefix
Skills	79	Auto-activated by context keywords
Hooks	34	Event-driven automation and context injection
Workflows	25+	Pre-defined process sequences
Subagents	24+	Specialized AI agents for specific tasks

Automatic Workflow Detection

Claude detects your intent from natural language and suggests the appropriate workflow:

What You Say	Detected Intent	Full Workflow Sequence
"I have a new feature idea"	Idea to PBI	`/idea` → `/refine` → `/story` → `/prioritize`
"Implement dark mode toggle"	Feature	`/plan` → `/plan-review` → `/cook` → `/code-simplifier` → `/code-review` → `/changelog` → `/test` → `/docs-update` → `/watzup`
"There's a bug in login"	Bug Fix	`/scout` → `/feature-investigation` → `/debug` → `/plan` → `/plan-review` → `/fix` → `/code-simplifier` → `/code-review` → `/changelog` → `/test`
"Refactor this service"	Refactor	`/plan` → `/plan-review` → `/code` → `/code-simplifier` → `/code-review` → `/test`
"Review this code"	Code Review	`/code-review` → `/watzup`
"Create tests for this PBI"	PBI to Tests	`/test-spec` → `/test-cases` → `/quality-gate`
"What's the sprint status?"	PM Reporting	`/status` → `/dependency`

What You Say	Detected Intent	Full Workflow Sequence
"Ready for release?"	Release Prep	`/quality-gate` → `/status`
"Sprint planning"	Sprint Planning	`/prioritize` → `/dependency` → `/team-sync`
"Ready for dev?"	Pre-Development	`/quality-gate` → `/plan`

Role-by-Role Guide

Product Owner (PO)

Primary Commands: `/idea`, `/refine`, `/prioritize`

Your AI-Assisted Workflow:

```
1. CAPTURE: /idea "Allow employees to upload profile photos"
   └── Creates: team-artifacts/ideas/260120-po-idea-employee-photo-upload.md

2. REFINE: /refine team-artifacts/ideas/260120-po-idea-employee-photo-upload.md
   └── Creates: team-artifacts/pbis/260120-pbi-employee-photo-upload.md
   └── Claude generates GIVEN/WHEN/THEN acceptance criteria

3. PRIORITIZE: /prioritize rice
   └── Scores all PBIs using RICE framework
   └── Orders backlog by calculated priority
```

What Claude Does for You:

- Structures raw ideas into proper problem/solution format
- Generates acceptance criteria automatically
- Calculates RICE/MoSCoW scores based on your input
- Identifies dependencies and risks

Quick Start:

```
# Your first command - capture an idea
/idea "Dark mode toggle for settings page"
```

Business Analyst (BA)

Primary Commands: `/refine` , `/story`

Your AI-Assisted Workflow:

```
1. REFINE (from PO idea): /refine team-artifacts/ideas/260120-po-idea-dark-mode.md
   └── Creates: team-artifacts/pbis/260120-pbi-dark-mode.md
   └── Generates 3+ acceptance criteria scenarios

2. STORY: /story team-artifacts/pbis/260120-pbi-dark-mode.md
   └── Creates: team-artifacts/pbis/stories/260120-us-dark-mode-*.md
   └── Breaks PBI into INVEST-compliant user stories
```

What Claude Does for You:

- Transforms vague requirements into structured PBIs
- Generates GIVEN/WHEN/THEN scenarios (happy path, edge cases, error cases)
- Slices PBIs into vertical user stories meeting INVEST criteria
- Identifies out-of-scope items automatically

Example Output - Acceptance Criteria:

```
Scenario: Successful theme toggle
  Given user is on the settings page
  And current theme is "light"
  When user clicks the dark mode toggle
  Then theme changes to "dark" within 200ms
  And preference is persisted to user profile
  And all UI components reflect the new theme

Scenario: Theme persists across sessions
  Given user has set theme to "dark"
  And user logs out and logs back in
  Then theme should still be "dark"
```

UX Designer

Primary Commands: `/design-spec`

Your AI-Assisted Workflow:

```
1. DESIGN SPEC: /design-spec team-artifacts/pbis/260120-pbi-dark-mode.md
   └── Creates: team-artifacts/design-specs/260120-designspec-dark-mode.md
   └── Component inventory with all states documented

2. FIGMA INTEGRATION: /design-spec figma.com/file/abc123/design
   └── Extracts design tokens from Figma MCP
   └── Maps to existing design system tokens
```

What Claude Does for You:

- Creates component inventory from requirements
- Documents all states (default, hover, active, disabled, error, loading)
- Maps designs to existing design system tokens
- Generates BEM class names following project conventions
- Includes accessibility requirements (WCAG 2.1)

Design Spec Structure:

```
## Component: theme-toggle

### States
| State | Visual | Interaction |
|-------|--------|-------------|
| Default (light) | Sun icon, light bg | Clickable |
| Default (dark) | Moon icon, dark bg | Clickable |
| Hover | Scale 1.05 | Cursor pointer |
| Disabled | 50% opacity | Not clickable |

### Design Tokens
- `--toggle-bg-light`: $gray-100
- `--toggle-bg-dark`: $gray-900
- `--toggle-icon-size`: 24px
- `--toggle-transition`: 200ms ease-out

### BEM Classes
- `.theme-toggle`
- `.theme-toggle__icon`
- `.theme-toggle__icon.--light`
- `.theme-toggle__icon.--dark`
- `.theme-toggle.--disabled`
```

Developer

Primary Commands: `/cook` , `/code` , `/fix` , `/plan` , `/test`

Your AI-Assisted Workflow:

Feature Implementation:

```
1. PLAN: /plan "implement dark mode toggle"
   └── Creates: plans/260120-dark-mode/plan.md
   └── Claude researches codebase, identifies patterns

2. REVIEW PLAN: /plan-review
   └── AI self-reviews the plan for validity
   └── Outputs: PASS / WARN / FAIL

3. IMPLEMENT: /cook plans/260120-dark-mode
   └── Step-by-step implementation with tests
   └── Follows platform patterns automatically
```

Bug Fix (Full Workflow):

```
1. SCOUT: /scout "where is theme handling"
   └── Finds relevant files and patterns

2. INVESTIGATE: /feature-investigation "theme persistence logic"
   └── Deep dive into the specific issue

3. DEBUG: /debug "dark mode not persisting"
   └── Root cause analysis with evidence

4. PLAN: /plan "fix theme persistence"
   └── Creates fix plan with approach

5. PLAN REVIEW: /plan-review
   └── Validates the fix approach

6. FIX: /fix
   └── Implements the fix following the plan

7. SIMPLIFY: /code-simplifier
   └── Cleans up the fix code

8. REVIEW: /code-review
   └── Self-reviews the changes

9. CHANGELOG: /changelog
   └── Documents the fix

10. TEST: /test
    └── Verifies fix doesn't break anything
```

What Claude Does for You:

- Searches codebase for existing patterns before writing new code
- Follows platform-specific patterns (Easy.Platform, Angular)
- Generates tests alongside implementation
- Respects architectural boundaries (service layers, repositories)
- Creates git commits with proper conventional commit format

Workflow Detection Examples:

```
"implement user photo upload" → Feature workflow
"fix the login bug" → Bug fix workflow
"refactor this service" → Refactoring workflow
"how does validation work" → Investigation workflow
```

QA Engineer

Primary Commands: `/test-spec` , `/test-cases`

Your AI-Assisted Workflow:

```
1. TEST SPEC: /test-spec team-artifacts/pbis/260120-pbi-dark-mode.md
   └── Creates: team-artifacts/test-specs/260120-testspec-dark-mode.md
   └── Test strategy with categories (positive, negative, edge)

2. TEST CASES: /test-cases team-artifacts/test-specs/260120-testspec-dark-mode.md
   └── Expands spec into detailed TC-{MOD}-{NNN} cases
   └── Each case includes Evidence field linking to code
```

What Claude Does for You:

- Generates test specifications from acceptance criteria
- Creates detailed test cases with TC-ID format (TC-GRO-001)
- Links each test case to implementation code (Evidence field)
- Identifies test categories: positive, negative, edge, performance
- Suggests automation approach (unit, integration, E2E)

Test Case Output Format:

```
### TC-GRO-001: Dark mode toggle activates correctly

**Category:** Positive
**Priority:** High

**Preconditions:**
- User is logged in
- Theme is currently "light"

**Steps:**
1. Navigate to Settings page
2. Locate theme toggle component
3. Click the toggle

**Expected Result:**
- Theme changes to "dark" within 200ms
- Toggle icon changes from sun to moon
- Local storage updated with theme preference

**Evidence:** `ThemeToggleComponent.ts:42` - `toggleTheme()` method
```

QC Specialist

Primary Commands: `/quality-gate`

Your AI-Assisted Workflow:

```
1. PRE-DEV GATE: /quality-gate pre-dev team-artifacts/pbis/260120-pbi-dark-mode.md
   └── Verifies: Acceptance criteria, dependencies, design specs

2. PRE-QA GATE: /quality-gate pre-qa team-artifacts/test-specs/260120-testspec-dark-mode.md
   └── Verifies: Test coverage, TC-ID format, Evidence fields

3. PRE-RELEASE GATE: /quality-gate pre-release PR#123
   └── Verifies: All tests pass, documentation updated, no blockers
```

What Claude Does for You:

- Runs standardized checklists at each gate

- Generates PASS/FAIL reports with specific issues
- Tracks quality metrics across sprints
- Ensures process compliance before handoffs

Gate Types:

Gate	When	Checks
`pre-dev`	Before development starts	Requirements clarity, design availability, dependencies
`pre-qa`	Before QA testing	Test coverage, TC format, evidence links
`pre-release`	Before deployment	All tests pass, docs updated, no blockers

Project Manager (PM)

Primary Commands: `/status`, `/dependency`, `/team-sync`

Your AI-Assisted Workflow:

```
1. STATUS: /status sprint
   └── Creates: plans/reports/260120-status-sprint.md
   └── Aggregates PBIs, commits, blockers

2. DEPENDENCIES: /dependency all
   └── Visualizes upstream/downstream dependencies
   └── Highlights blockers and risks

3. MEETING PREP: /team-sync daily
   └── Generates standup agenda with yesterday/today/blockers
```

What Claude Does for You:

- Generates status reports from git activity and artifacts
- Maps dependencies between features and teams
- Creates meeting agendas for daily/weekly/sprint ceremonies
- Identifies blockers and risks automatically

Report Types:

Command	Output	Use Case
`/status sprint`	Sprint progress report	Daily status
`/status project`	Full project report	Weekly updates
`/status feature-{name}`	Feature-specific report	Milestone tracking
`/team-sync daily`	Standup agenda	Daily standup
`/team-sync sprint-review`	Sprint review agenda	Sprint end

Delivery Manager (DM)

Primary Commands: `/status project` , `/dependency` , `/quality-gate pre-release`

Your AI-Assisted Workflow:

```
1. RELEASE READINESS: /quality-gate pre-release PR#456
   └── Verifies all quality gates passed
   └── Checks documentation completeness
   └── Confirms no critical blockers

2. CROSS-TEAM DEPENDENCIES: /dependency all
   └── Maps dependencies across squads/teams
   └── Identifies critical path items
   └── Highlights external blockers

3. STAKEHOLDER REPORTING: /status project
   └── Comprehensive project health report
   └── Milestone progress tracking
   └── Risk and blocker summary
```

What Claude Does for You:

- Aggregates status across multiple teams/squads
- Tracks release readiness across all quality gates
- Identifies cross-team dependencies and blockers
- Generates stakeholder-ready reports

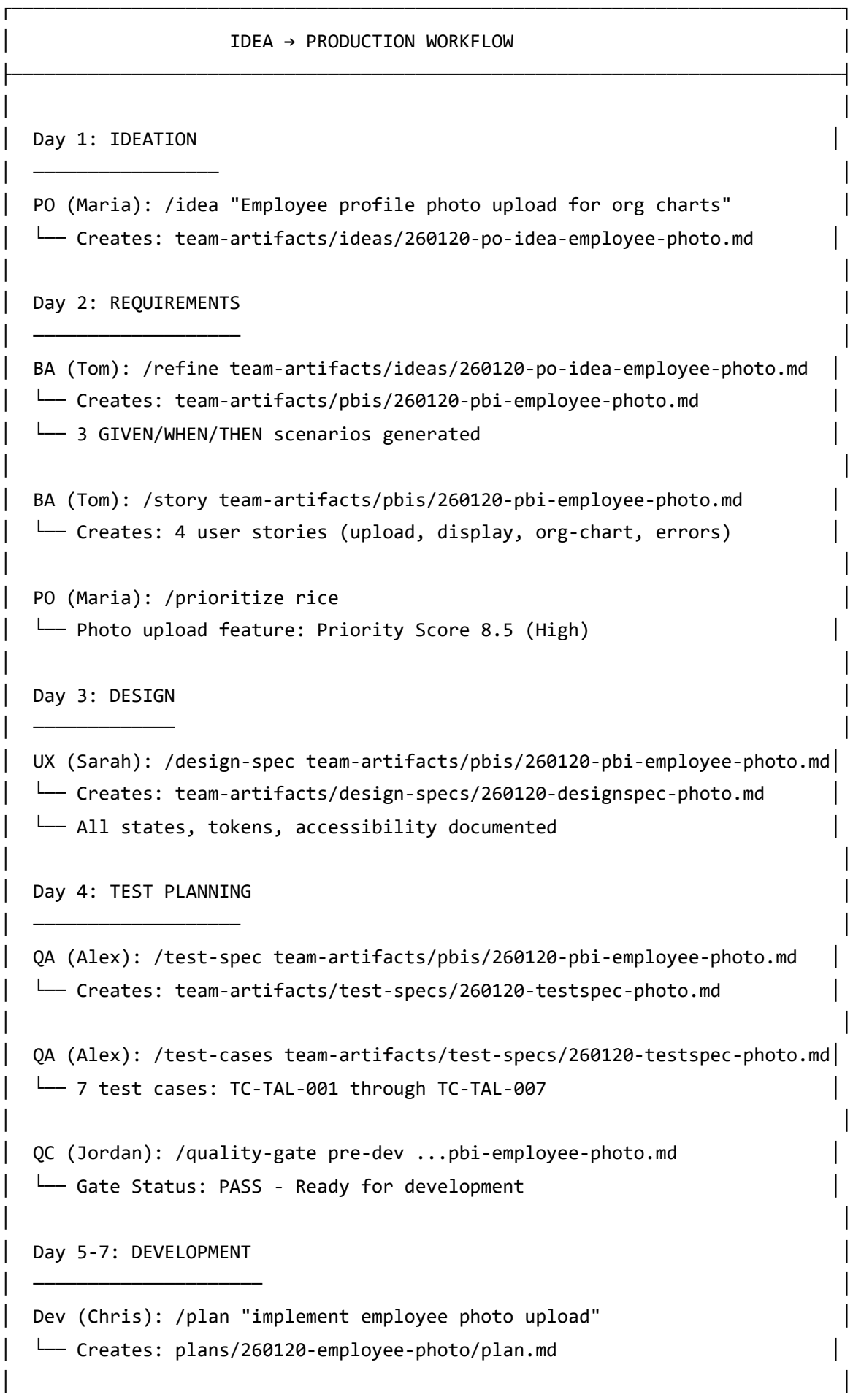
Key Reports:

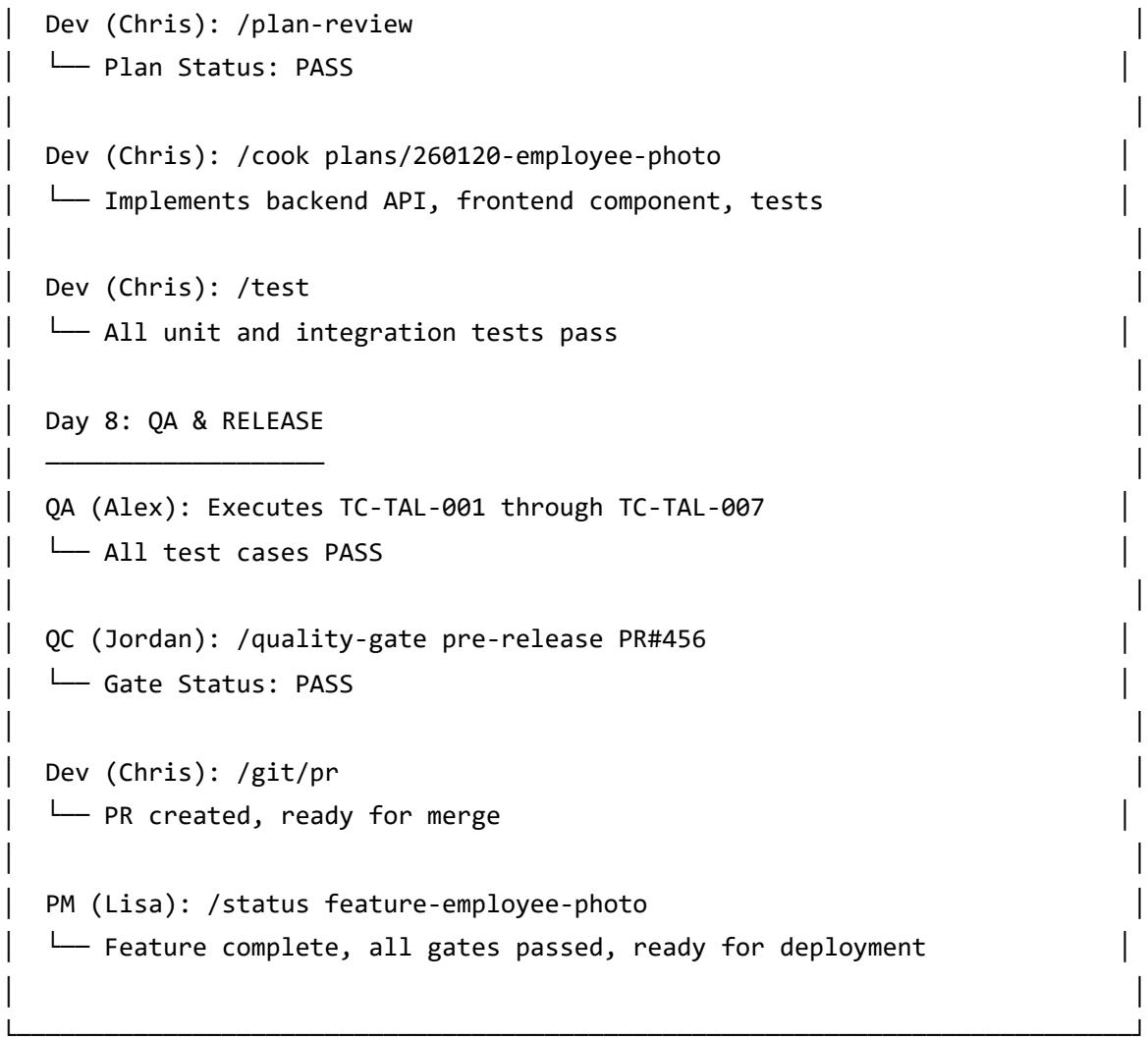
Command	Output	Audience
`/status project`	Full project report	Stakeholders
`/dependency all`	Dependency map	Technical leads
`/quality-gate pre-release`	Release checklist	Release team

Complete SDLC Workflow

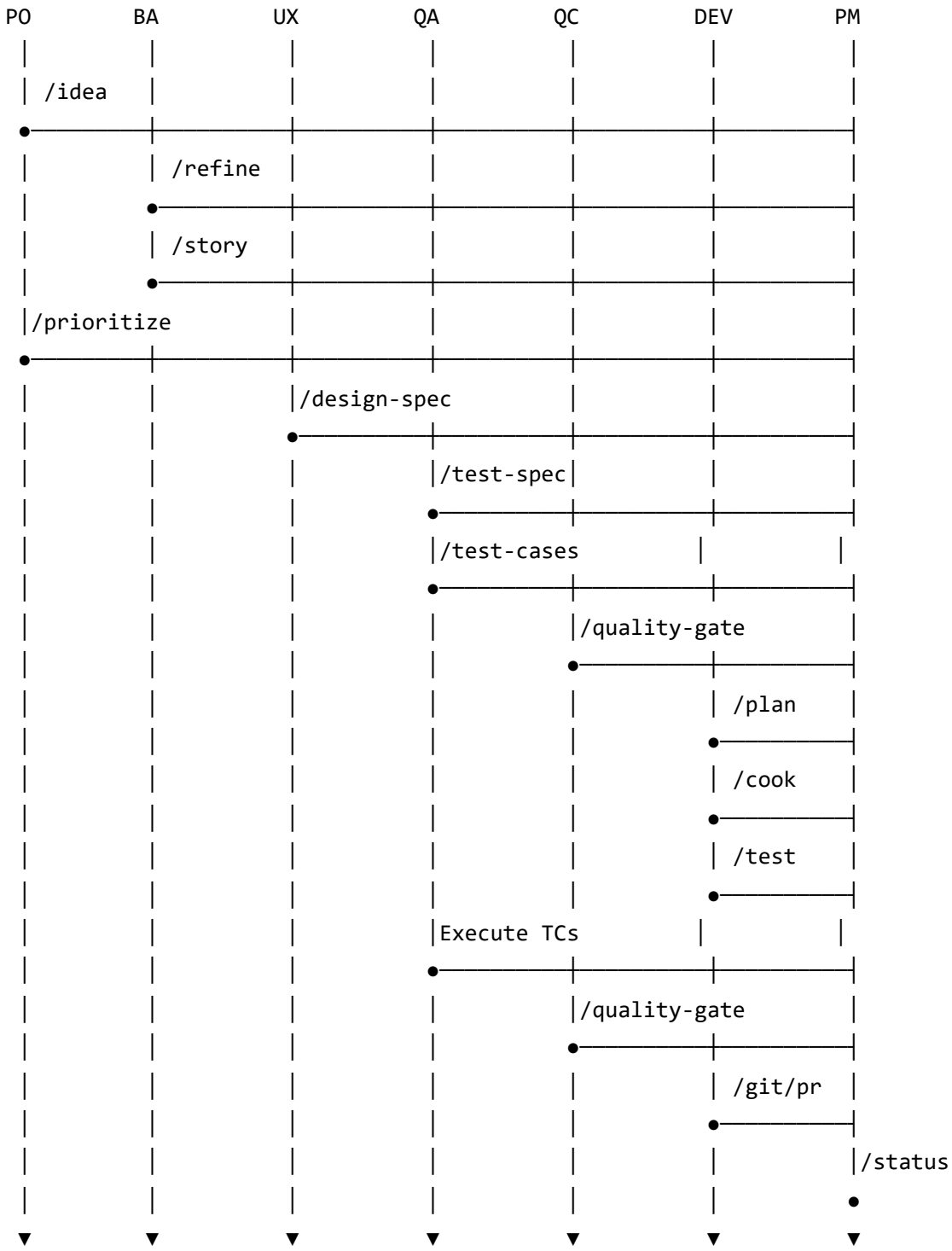
End-to-End Example: Employee Photo Upload Feature

This example shows how a feature flows through all roles using Claude Code.


IDEA → PRODUCTION WORKFLOW
Day 1: IDEATION
PO (Maria): /idea "Employee profile photo upload for org charts"
└── Creates: team-artifacts/ideas/260120-po-idea-employee-photo.md
Day 2: REQUIREMENTS
BA (Tom): /refine team-artifacts/ideas/260120-po-idea-employee-photo.md
└── Creates: team-artifacts/pbis/260120-pbi-employee-photo.md
└── 3 GIVEN/WHEN/THEN scenarios generated
BA (Tom): /story team-artifacts/pbis/260120-pbi-employee-photo.md
└── Creates: 4 user stories (upload, display, org-chart, errors)
PO (Maria): /prioritize rice
└── Photo upload feature: Priority Score 8.5 (High)
Day 3: DESIGN
UX (Sarah): /design-spec team-artifacts/pbis/260120-pbi-employee-photo.md
└── Creates: team-artifacts/design-specs/260120-designspec-photo.md
└── All states, tokens, accessibility documented
Day 4: TEST PLANNING
QA (Alex): /test-spec team-artifacts/pbis/260120-pbi-employee-photo.md
└── Creates: team-artifacts/test-specs/260120-testspec-photo.md
QA (Alex): /test-cases team-artifacts/test-specs/260120-testspec-photo.md
└── 7 test cases: TC-TAL-001 through TC-TAL-007
QC (Jordan): /quality-gate pre-dev ...pbi-employee-photo.md
└── Gate Status: PASS - Ready for development
Day 5-7: DEVELOPMENT
Dev (Chris): /plan "implement employee photo upload"
└── Creates: plans/260120-employee-photo/plan.md


Dev (Chris): /plan-review
└── Plan Status: PASS
Dev (Chris): /cook plans/260120-employee-photo
└── Implements backend API, frontend component, tests
Dev (Chris): /test
└── All unit and integration tests pass
Day 8: QA & RELEASE
QA (Alex): Executes TC-TAL-001 through TC-TAL-007
└── All test cases PASS
QC (Jordan): /quality-gate pre-release PR#456
└── Gate Status: PASS
Dev (Chris): /git/pr
└── PR created, ready for merge
PM (Lisa): /status feature-employee-photo
└── Feature complete, all gates passed, ready for deployment

Swimlane Diagram



Example Scenarios

Scenario 1: Bug Fix Workflow

Context: QA finds a bug where uploaded photos don't display in the org chart.

Full Workflow: scout → investigate → debug → plan → plan-review → fix → code-simplifier → code-review → changelog → test

```
# QA reports the bug
QA: "Bug found - employee photo not showing in org chart after upload"

# 1. SCOUT - Find relevant files
Dev: /scout "where is org chart photo rendering"
└── Claude finds: OrgChartComponent.ts, EmployeeAvatarComponent.ts, OrgChartService.cs

# 2. INVESTIGATE - Deep dive into the issue
Dev: /feature-investigation "org chart photo loading logic"
└── Claude traces data flow: API → Service → Component

# 3. DEBUG - Root cause analysis
Dev: /debug "photo not showing in org chart"
└── Root cause: Missing navigation property loading for Photo entity

# 4. PLAN - Create fix plan
Dev: /plan "fix org chart photo loading"
└── Creates: plans/260120-orgchart-photo-fix/plan.md

# 5. PLAN REVIEW - Validate approach
Dev: /plan-review
└── Plan Status: PASS - approach is correct

# 6. FIX - Implement the fix
Dev: /fix
└── Claude:
    1. Adds loadRelatedEntities: e => e.Photo in OrgChartService.cs
    2. Updates component to handle photo URL
    3. Writes unit test

# 7. CODE SIMPLIFIER - Clean up
Dev: /code-simplifier
└── Optimizes the fix code

# 8. CODE REVIEW - Self-review
Dev: /code-review
└── Review: PASS - follows platform patterns

# 9. CHANGELOG - Document fix
Dev: /changelog
└── Adds entry: "fix: load employee photo in org chart query"

# 10. TEST - Verify fix
```

```
Dev: /test
└── All tests pass

# Create PR
Dev: /git/pr
└── PR #457 created with full context
```

Scenario 2: Sprint Planning

Context: PM preparing for sprint planning ceremony.

```
# Generate backlog status
PM: /status sprint
└── Creates sprint report with:
    - 15 PBIs in backlog
    - 8 in progress
    - 3 blocked

# Check dependencies
PM: /dependency all
└── Visualizes:
    - Photo upload → Org chart update
    - Theme toggle → No dependencies
    - Email service → Blocked by infrastructure

# Prepare meeting
PM: /team-sync sprint-planning
└── Generates agenda:
    1. Review sprint goal
    2. Capacity planning
    3. Backlog prioritization
    4. Dependency resolution
    5. Commitment

# PO prioritizes
PO: /prioritize rice scope:sprint
└── Orders backlog by RICE score
```

Scenario 3: Documentation Update

Context: Developer completes a feature and needs to update docs.

```
# After implementation
Dev: /docs/update

Claude:
1. Detects recent changes (git diff)
2. Identifies affected documentation
3. Updates:
   - API endpoint docs
   - Component usage examples
   - Configuration references
4. Creates changelog entry

# Generate release notes
Dev: /release-notes
└── Generates markdown release notes from recent commits
```

Complete Workflow Reference

All 25 workflows defined in `workflows.json` with their full sequences:

Core Development Workflows

Workflow	Trigger Example	Full Sequence
Feature	"implement dark mode"	plan → plan-review → cook → code-simplifier → code-review → changelog → test → docs-update → watzup
Bug Fix	"fix login error"	scout → investigate → debug → plan → plan-review → fix → code-simplifier → code-review → changelog → test
Refactor	"refactor this service"	plan → plan-review → code → code-simplifier → code-review → test
Investigation	"how does auth work"	scout → investigate
Code Review	"review my changes"	code-review → watzup

Workflow	Trigger Example	Full Sequence
Testing	"run tests"	test
Batch Operation	"rename across all files"	plan → plan-review → code → test

Requirements & Planning Workflows

Workflow	Trigger Example	Full Sequence
Idea to PBI	"new feature idea"	idea → refine → story → prioritize
Sprint Planning	"sprint planning"	prioritize → dependency → team-sync
Pre-Development	"ready for dev?"	quality-gate → plan

Quality & Testing Workflows

Workflow	Trigger Example	Full Sequence
PBI to Tests	"create tests for this PBI"	test-spec → test-cases → quality-gate
Quality Assurance	"run quality gate"	quality-gate
Release Prep	"ready for release?"	quality-gate → status

Design Workflows

Workflow	Trigger Example	Full Sequence
Figma Design	"figma.com/file/..."	design-spec
Design Workflow	"design this component"	design-spec → code-review

Documentation Workflows

Workflow	Trigger Example	Full Sequence
Documentation	"update docs"	scout → investigate → docs-update → watzup

Workflow	Trigger Example	Full Sequence
Business Feature Docs	"document this feature"	scout → investigate → docs-update → watzup

PM & Team Workflows

Workflow	Trigger Example	Full Sequence
PM Reporting	"sprint status"	status → dependency
Team Ceremonies	"daily standup"	team-sync
Dependency Analysis	"what's blocking?"	dependency

Direct Command Workflows

These workflows invoke single commands directly:

Workflow	Command	Trigger Example
Direct: Idea	/idea	"I have an idea"
Direct: Refine	/refine	"create a PBI"
Direct: Story	/story	"create user stories"
Direct: Design Spec	/design-spec	"create design spec"
Direct: Test Spec	/test-spec	"create test spec"
Direct: Test Cases	/test-cases	"generate test cases"
Direct: Dependency	/dependency	"show dependencies"
Direct: Prioritize	/prioritize	"prioritize backlog"
Direct: Status	/status	"status report"
Direct: Team Sync	/team-sync	"standup agenda"
Direct: Quality Gate	/quality-gate	"quality check"

Command Quick Reference

By Role

Role	Primary Commands	When to Use
PO	`/idea`, `/prioritize`	Capturing ideas, ordering backlog
BA	`/refine`, `/story`	Requirements, user stories
UX	`/design-spec`	Design specifications
QA	`/test-spec`, `/test-cases`	Test planning
QC	`/quality-gate`	Quality checkpoints
Dev	`/plan`, `/cook`, `/fix`, `/test`	Implementation
PM	`/status`, `/dependency`, `/team-sync`	Reporting, tracking

Command Cheat Sheet

```
CAPTURE & REFINE
├── /idea "description"              # Capture new idea
├── /refine {idea-file}              # Idea → PBI
├── /story {pbi-file}                # PBI → User Stories
└── /prioritize rice|moscow          # Order backlog

DESIGN & TEST
├── /design-spec {source}            # Create design spec
├── /test-spec {pbi-file}            # Generate test spec
├── /test-cases {spec-file}          # Expand to test cases
└── /quality-gate pre-dev|pre-qa|pre-release  # Quality checkpoint

DEVELOP
├── /plan "description"              # Create implementation plan
├── /plan-review                     # Auto-review plan
├── /cook {plan-dir}                 # Implement feature
├── /fix "description"               # Fix bug
├── /code                            # Execute existing plan
└── /test                            # Run tests

GIT & REVIEW
├── /git/cm                          # Commit changes
├── /git/pr                          # Create pull request
├── /review                          # Code review
└── /review/codebase                 # Full audit

INVESTIGATE
├── /scout "query"                   # Quick codebase scan
├── /feature-investigation "query"   # Deep investigation
└── /debug "issue"                   # Debug issue

PM REPORTING
├── /status sprint|project           # Status report
├── /dependency all|{target}         # Dependency map
└── /team-sync daily|weekly|sprint-review  # Meeting agenda

UTILITY
├── /watzup                          # Current status
├── /checkpoint "description"        # Save progress
└── /recover                         # Restore from checkpoint
```

Adoption Roadmap

Week 1: Quick Wins (All Roles)

Day	Role	Action	Expected Outcome
1	All	Install Claude Code, run `/watzup`	Environment verified
2	PO	Capture 1 idea with `/idea`	First artifact created
2	BA	Refine 1 idea with `/refine`	First PBI generated
3	Dev	Investigate codebase with `/scout`	Familiar with exploration
3	QA	Generate test spec with `/test-spec`	First test artifact
4	PM	Generate status report `/status sprint`	First PM report
5	All	Team retrospective on Claude usage	Feedback collected

Week 2-4: Process Integration

Phase	Focus	Success Metric
Week 2	Use Claude for all new PBIs	100% PBIs have Claude-generated AC
Week 3	Integrate quality gates	All PRs pass `/quality-gate pre-release`
Week 4	Full workflow adoption	End-to-end feature uses Claude commands

Success Criteria

- ☐ Every team member has executed at least 3 commands
- ☐ 1 feature completed using full workflow (Idea → PBI → Dev → Test → Release)
- ☐ 1 bug fixed using `/scout` → `/debug` → `/fix` workflow
- ☐ Team feedback session completed with improvements documented

Measuring ROI

Track these metrics before/after adoption:

Metric	How to Measure	Target
PBI creation time	Time from idea to refined PBI	-50%
Test case generation	Time to create test spec	-70%
Bug investigation time	Time to root cause	-60%
Onboarding time	New member productivity	-40%

Setup & Prerequisites

System Requirements

Requirement	Minimum	Recommended
Claude Code CLI	Latest	Latest
Node.js	18+	20+
Git	2.x	2.40+
VS Code (optional)	1.85+	Latest

Installation Verification

```
# Check Claude Code
claude --version

# Check Node.js
node --version

# Check Git
git --version

# Check hooks are configured
ls .claude/hooks/
```

First-Time Setup

1. **Clone the repository**

   ```
   git clone https://github.com/bravocompany/bravosuite.git
   cd bravosuite
   ```

2. **Start Claude Code**

   ```
   claude
   ```

3. **Verify configuration**

   ```
   /watzup
   ```

4. **Try your first command** (based on your role)
   - PO: `/idea "test feature"`
   - BA: `/refine` (provide an idea file)
   - Dev: `/scout "how does authentication work"`
   - QA: `/test-spec` (provide a PBI file)

Troubleshooting

Common Issues

Issue	Cause	Solution
Command not found	Skill file missing	Run `ls .claude/skills/`
Workflow not detected	Keywords not matched	Use explicit command with `/`
Artifact path error	Wrong naming format	Use `{YYMMDD}-{role}-{type}-{slug}.md`
Quality gate fails	Missing criteria	Review gate report, fix issues
Context not injected	Hook not running	Check `.claude/settings.json`

Getting Help

1. **In Claude Code:** `/ck-help` - Lists all commands

2. **Documentation:** `docs/claude/` - Full reference
3. **Team Guide:** `docs/claude/team-collaboration-guide.md` - Role-specific workflows
4. **Quick Start:** `docs/claude/quick-start.md` - 5-minute onboarding

Appendix: Technical Architecture

System Components

```
.claude/
├── settings.json           # Main configuration (hooks, permissions)
├── .ck.json                # ClaudeKit config (coding levels, assertions)
├── workflows.json          # Workflow automation (25+ workflows, multilingual)
├── .mcp.json               # MCP server integrations
├── commands/               # 107 slash commands
│   ├── workflow/           # /cook, /plan, /code
│   ├── git/                # /git/cm, /git/pr
│   └── team/               # /idea, /refine, /story, etc.
├── skills/                 # 79 context-activated skills
│   ├── product-owner/      # PO skill
│   ├── business-analyst/   # BA skill
│   ├── qa-engineer/        # QA skill
│   └── ...
├── hooks/                  # 34 event hooks
│   ├── workflow-router.cjs # Auto-detects workflow intent
│   ├── pattern-learner.cjs # Learns from corrections
│   └── lib/                # 41 shared modules
├── agents/                 # 24+ subagent configurations
└── memory/                 # ACE learning data
    ├── playbook.json       # Active learned patterns
    └── events-stream.jsonl # Event history
```

Hook Event Lifecycle

```
User Input → SessionStart hooks
          → PrePrompt hooks (workflow detection, pattern injection)
          → Claude processes
          → PreToolUse hooks (validation, context injection)
          → Tool executes
          → PostToolUse hooks (event capture, tracking)
          → Response
          → PreCompact hooks (if context compaction needed)
          → SessionEnd hooks (cleanup, persist state)
```

Artifact Naming Convention

```
{YYMMDD}-{role}-{type}-{slug}.md

Examples:
260120-po-idea-employee-photo.md
260120-ba-pbi-employee-photo.md
260120-ux-designspec-photo-upload.md
260120-qa-testspec-photo-upload.md
260120-qc-gate-pre-dev-photo.md
```

MCP Server Integrations

Server	Purpose
`github`	Repository operations, PR management
`figma`	Design token extraction
`context7`	Documentation retrieval
`sequential-thinking`	Complex problem solving
`memory`	Knowledge graph persistence

Document History

Version	Date	Author	Changes
1.0	2026-01-20	Claude Code	Initial comprehensive guide

Questions? Contact the Engineering team or refer to docs/claude/README.md for detailed documentation.

Feedback: Report issues at https://github.com/anthropics/claude-code/issues