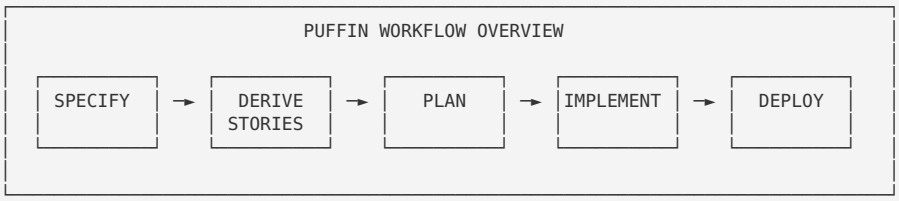
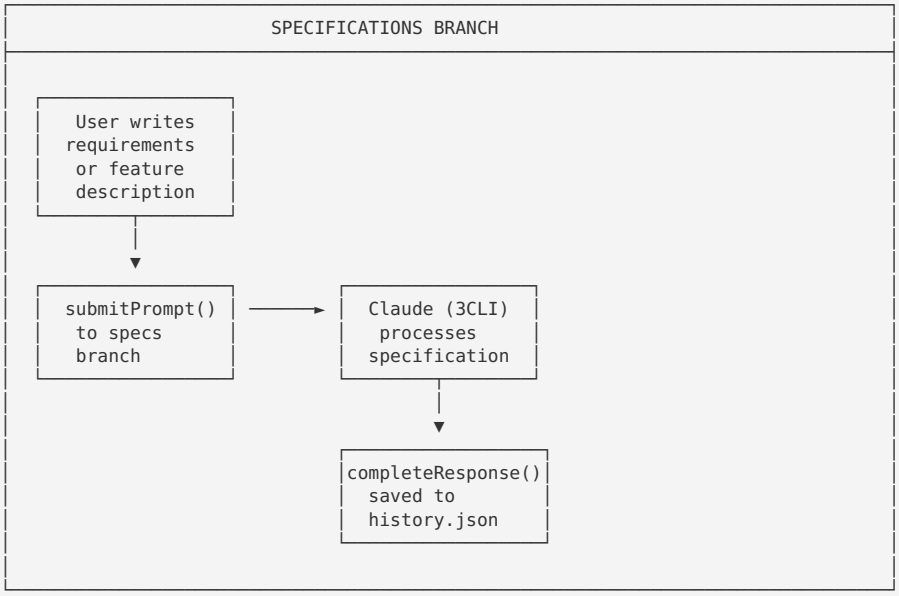
Puffin Information Flow Diagram

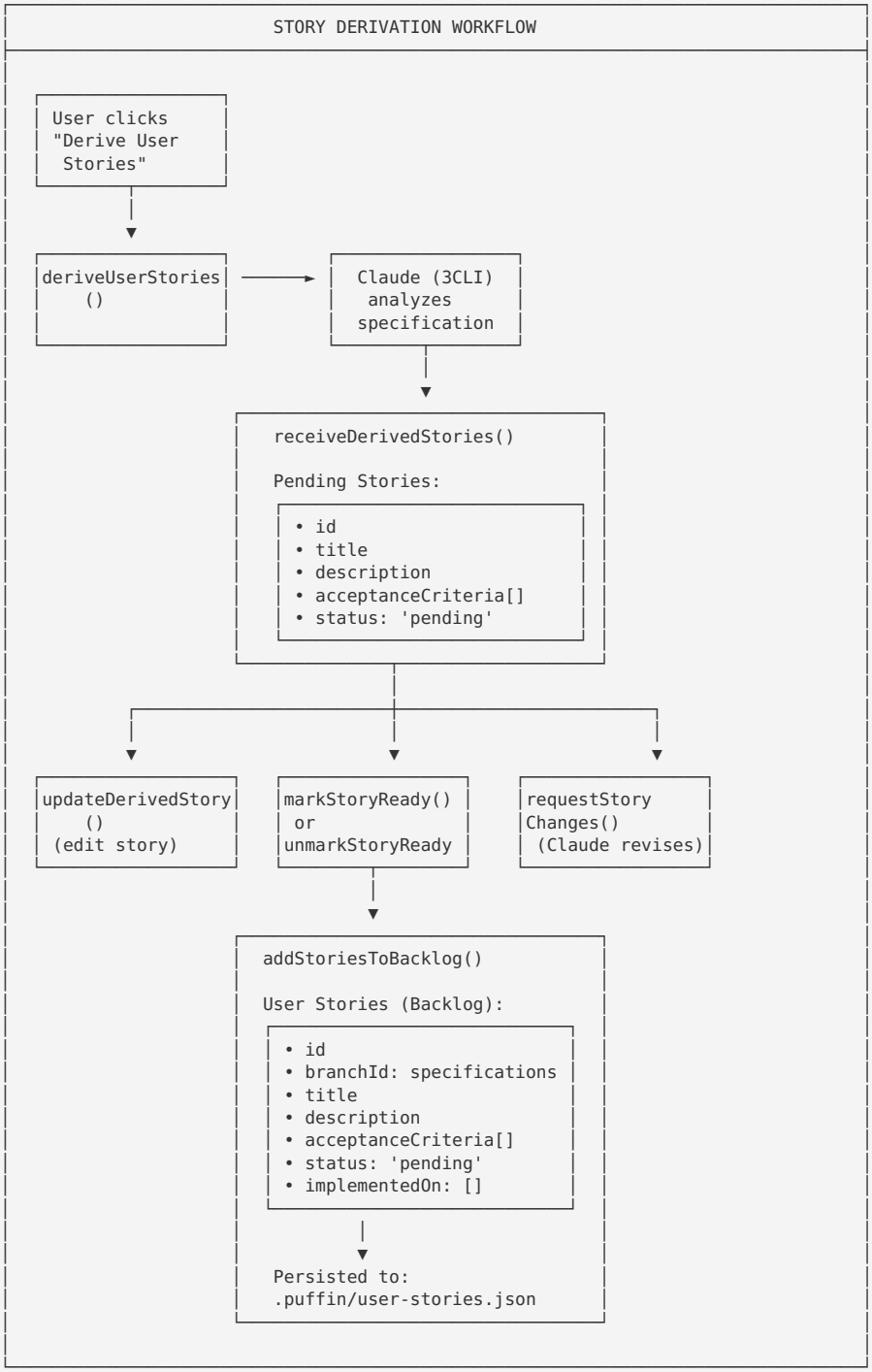
Overview: Specification to Deployment Pipeline



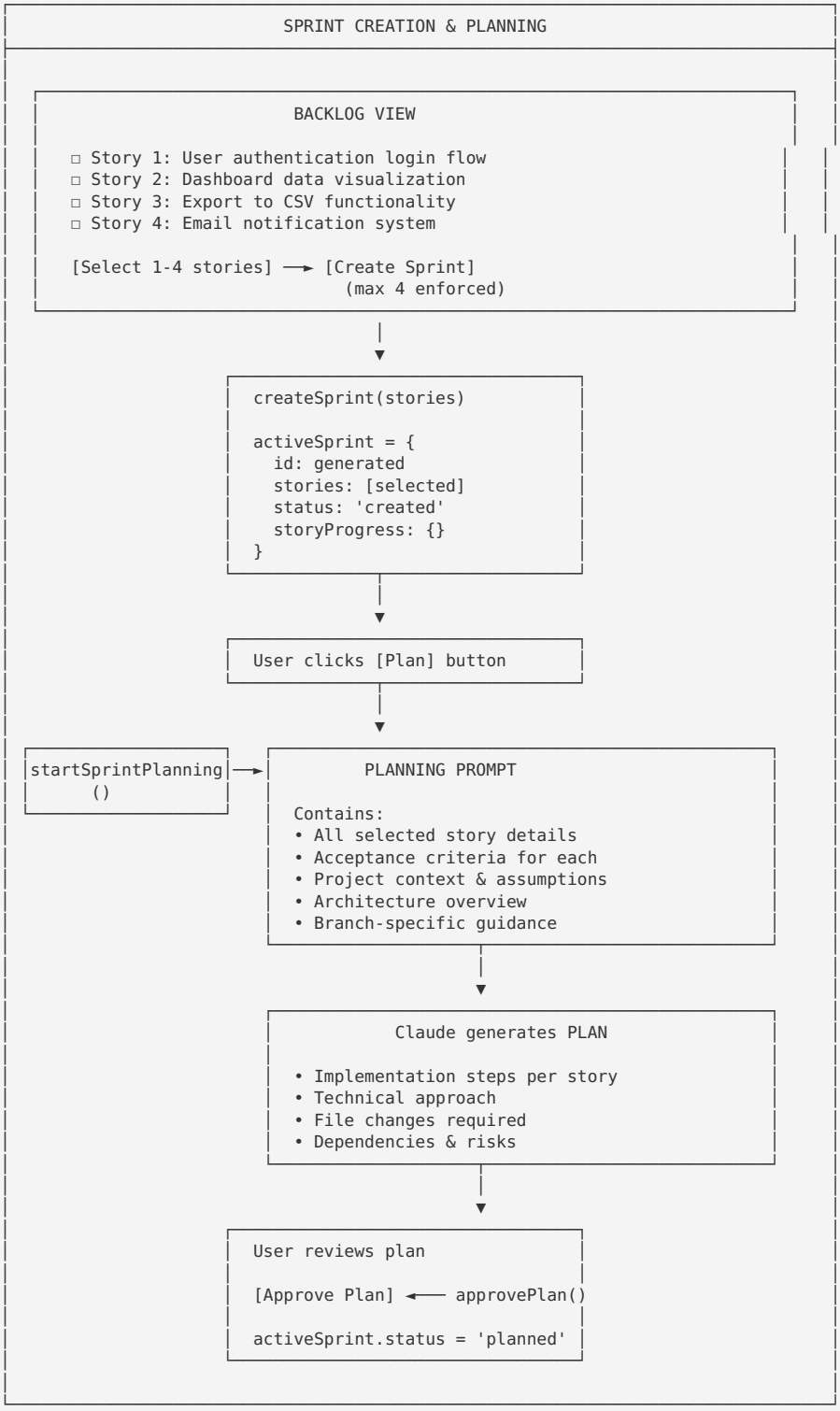
Phase 1: Specification & Requirements



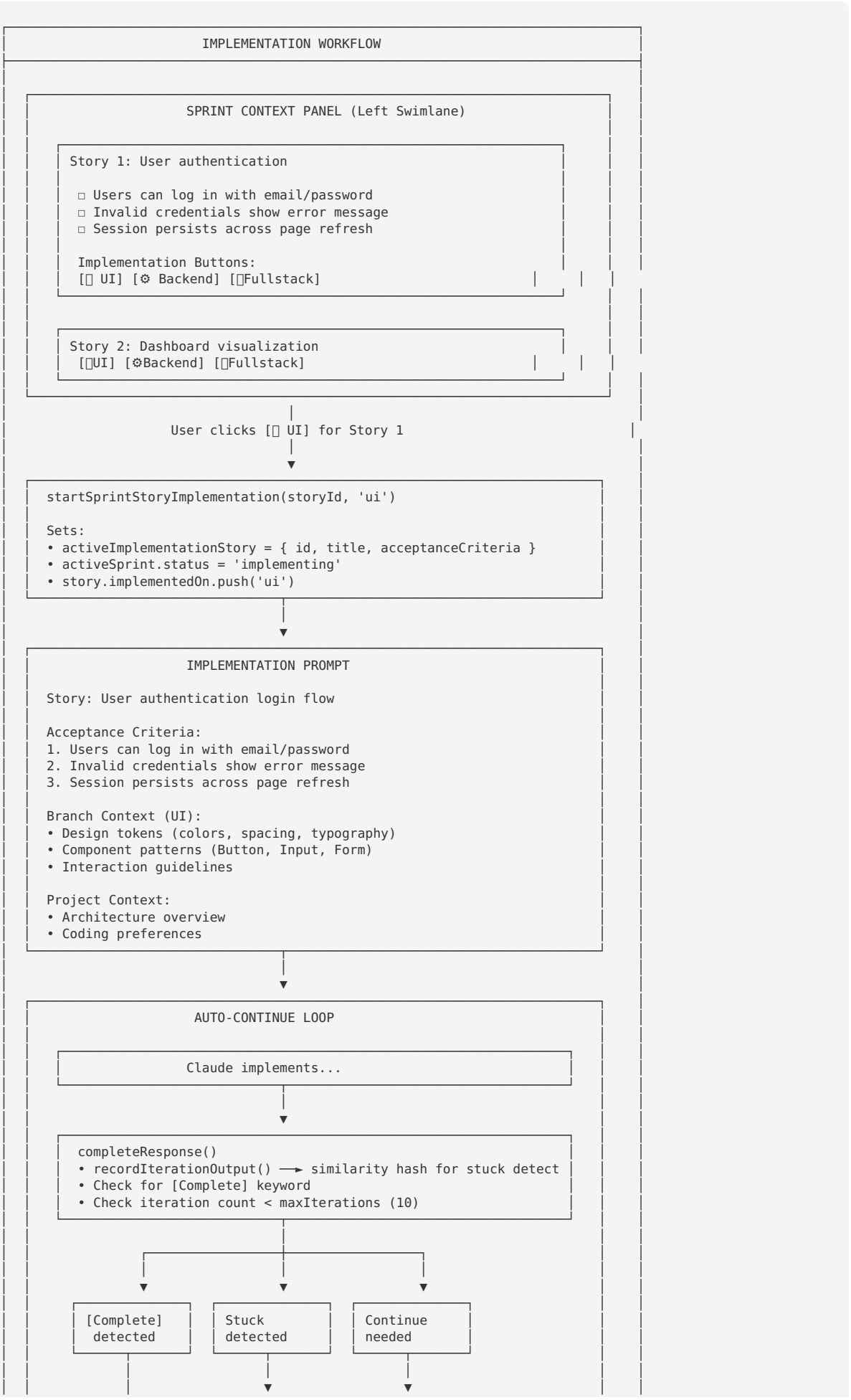
Phase 2: User Story Derivation

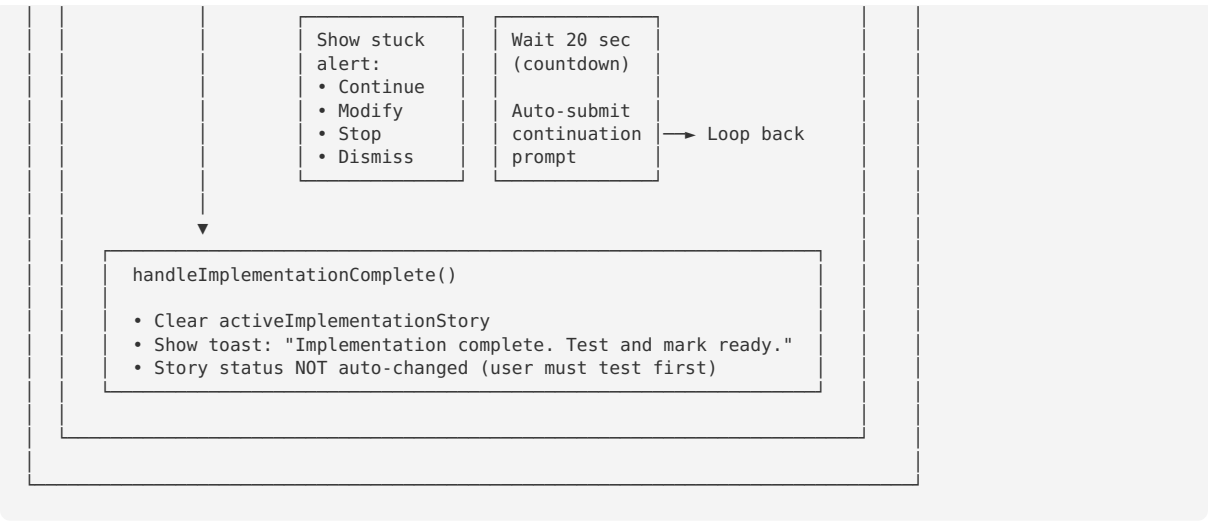


Phase 3: Sprint Planning

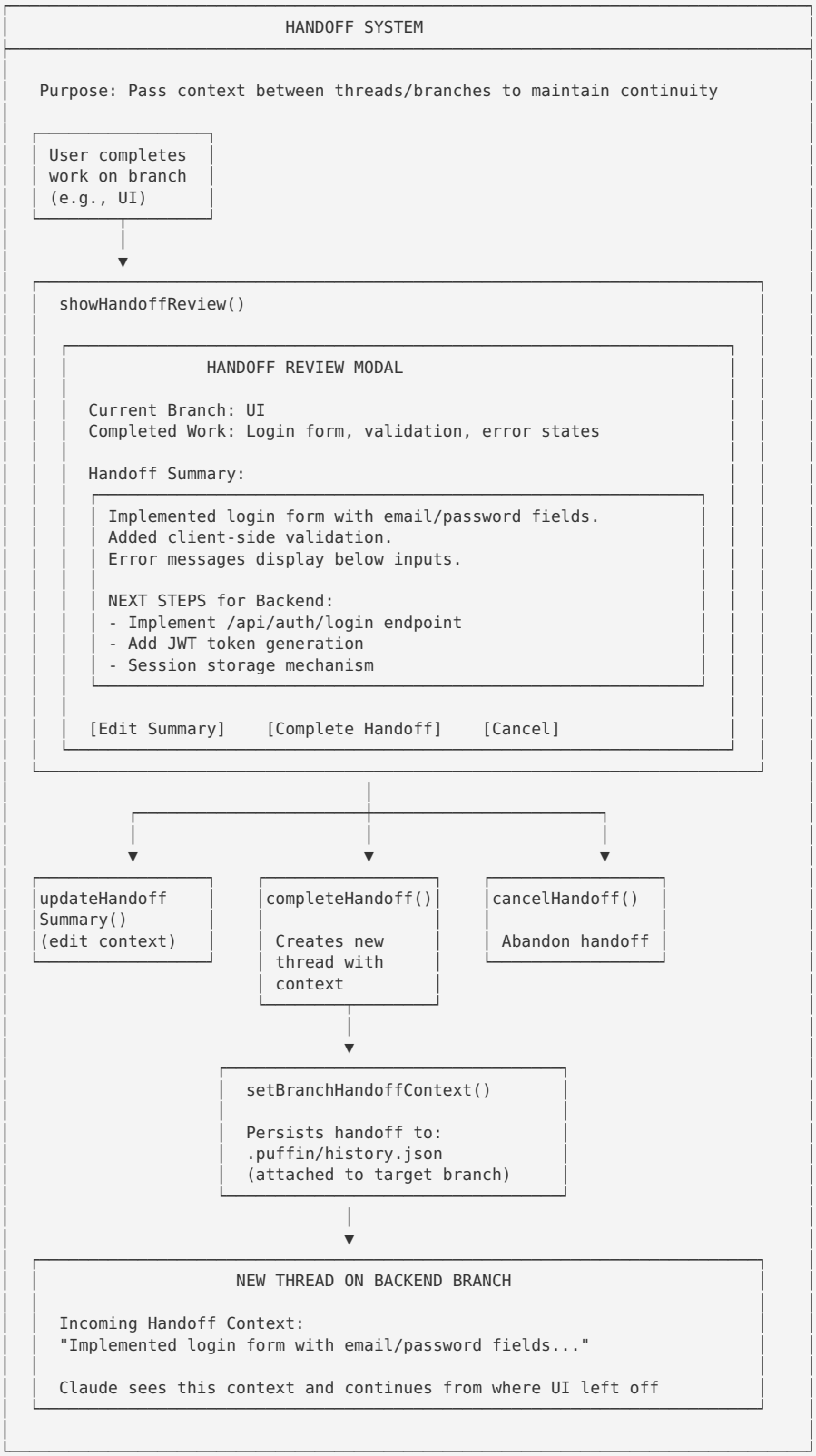


Phase 4: Implementation

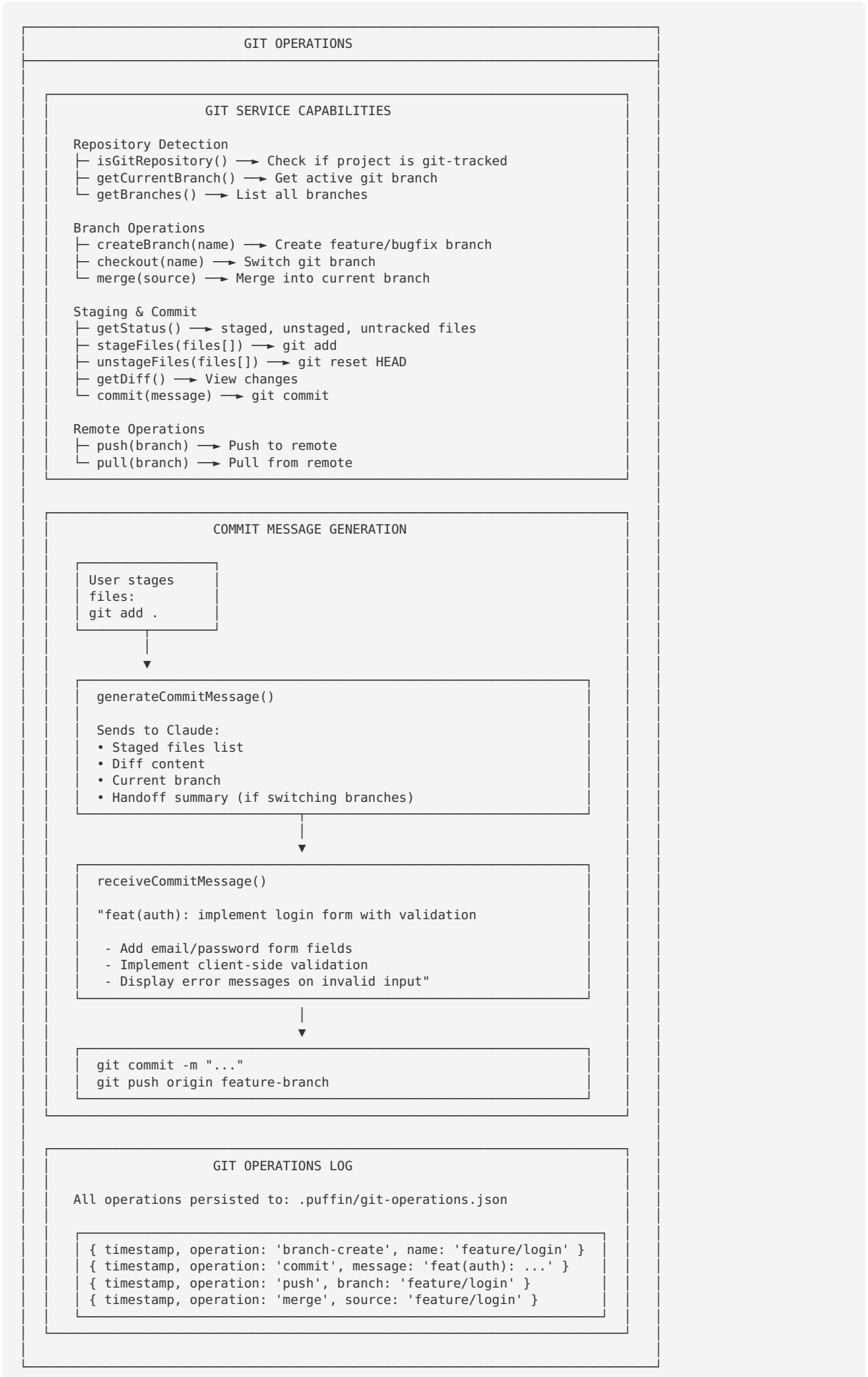

IMPLEMENTATION WORKFLOW
SPRINT CONTEXT PANEL (Left Swimlane)
Story 1: User authentication
□ Users can log in with email/password
□ Invalid credentials show error message
□ Session persists across page refresh
Implementation Buttons:
[□ UI] [⚙ Backend] [□Fullstack]
Story 2: Dashboard visualization
[□UI] [⚙Backend] [□Fullstack]
User clicks [□ UI] for Story 1
startSprintStoryImplementation(storyId, 'ui')
Sets:
• activeImplementationStory = { id, title, acceptanceCriteria }
• activeSprint.status = 'implementing'
• story.implementedOn.push('ui')
IMPLEMENTATION PROMPT
Story: User authentication login flow
Acceptance Criteria:
1. Users can log in with email/password
2. Invalid credentials show error message
3. Session persists across page refresh
Branch Context (UI):
• Design tokens (colors, spacing, typography)
• Component patterns (Button, Input, Form)
• Interaction guidelines
Project Context:
• Architecture overview
• Coding preferences
AUTO-CONTINUE LOOP
Claude implements...
completeResponse()
• recordIterationOutput() ──► similarity hash for stuck detect
• Check for [Complete] keyword
• Check iteration count < maxIterations (10)
[Complete] detected
Stuck detected
Continue needed


Show stuck alert:
• Continue
• Modify
• Stop
• Dismiss
Wait 20 sec (countdown)
Auto-submit continuation prompt
Loop back
handleImplementationComplete()
• Clear activeImplementationStory
• Show toast: "Implementation complete. Test and mark ready."
• Story status NOT auto-changed (user must test first)

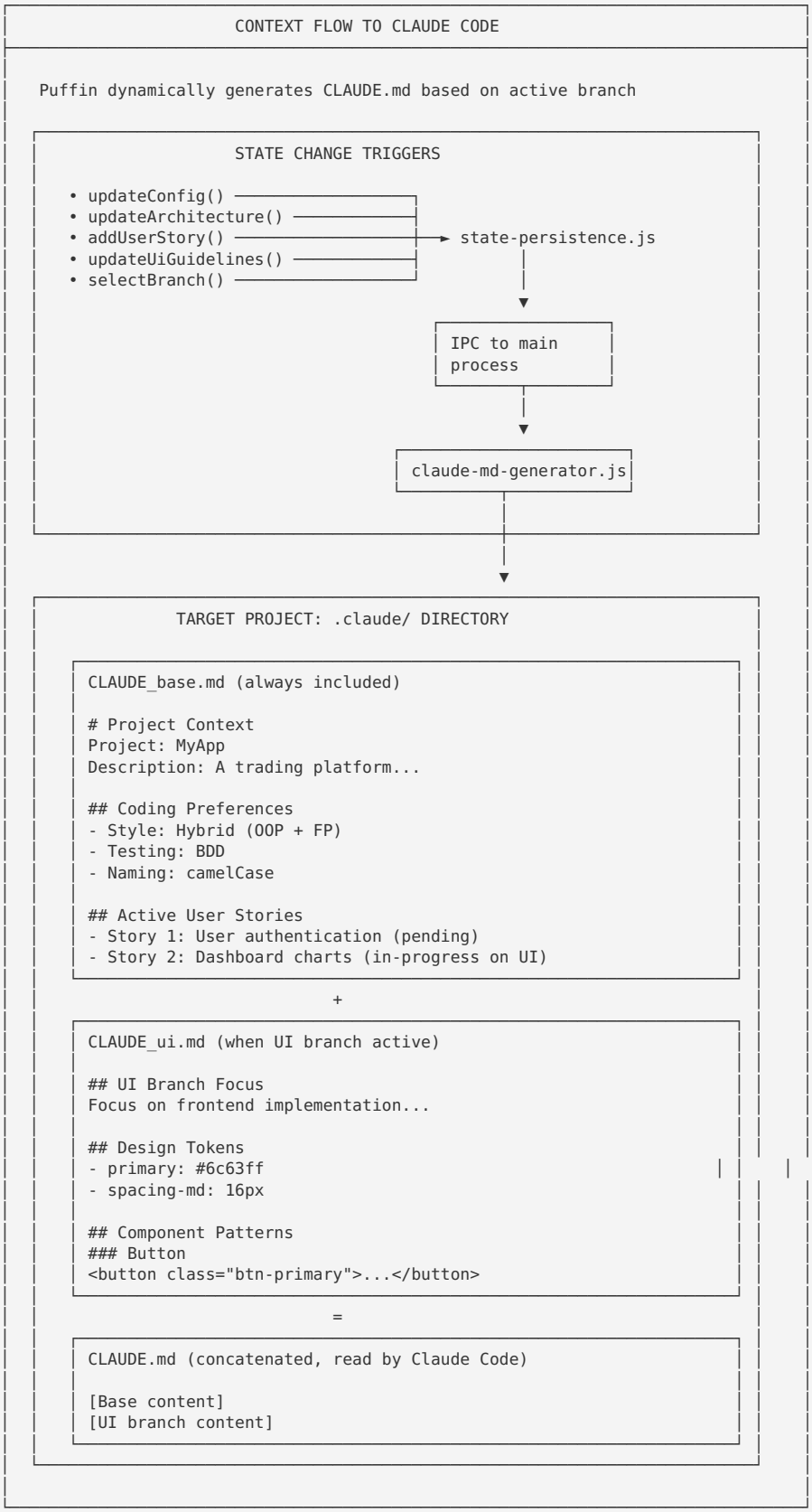
Phase 5: Handoff Between Branches



Phase 6: Git Operations & Deployment


GIT OPERATIONS
GIT SERVICE CAPABILITIES
Repository Detection
├─ isGitRepository() ──► Check if project is git-tracked
├─ getCurrentBranch() ──► Get active git branch
└─ getBranches() ──► List all branches
Branch Operations
├─ createBranch(name) ──► Create feature/bugfix branch
├─ checkout(name) ──► Switch git branch
└─ merge(source) ──► Merge into current branch
Staging & Commit
├─ getStatus() ──► staged, unstaged, untracked files
├─ stageFiles(files[]) ──► git add
├─ unstageFiles(files[]) ──► git reset HEAD
├─ getDiff() ──► View changes
└─ commit(message) ──► git commit
Remote Operations
├─ push(branch) ──► Push to remote
└─ pull(branch) ──► Pull from remote
COMMIT MESSAGE GENERATION
User stages files:
git add .
generateCommitMessage()
Sends to Claude:
• Staged files list
• Diff content
• Current branch
• Handoff summary (if switching branches)
receiveCommitMessage()
"feat(auth): implement login form with validation
- Add email/password form fields
- Implement client-side validation
- Display error messages on invalid input"
git commit -m "..."
git push origin feature-branch
GIT OPERATIONS LOG
All operations persisted to: .puffin/git-operations.json
{ timestamp, operation: 'branch-create', name: 'feature/login' }
{ timestamp, operation: 'commit', message: 'feat(auth): ...' }
{ timestamp, operation: 'push', branch: 'feature/login' }
{ timestamp, operation: 'merge', source: 'feature/login' }

Dynamic CLAUDE.md Generation

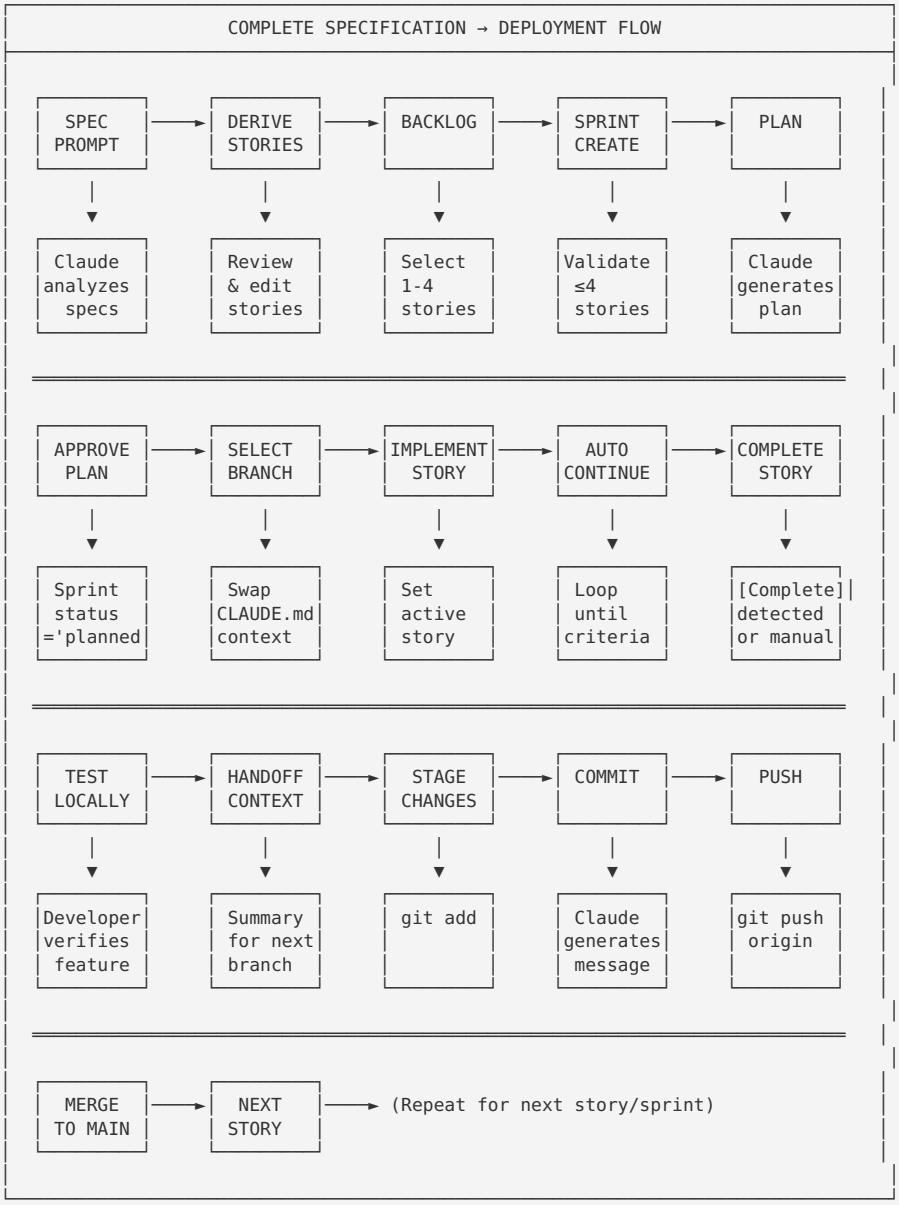


Branch Types & Context Summary

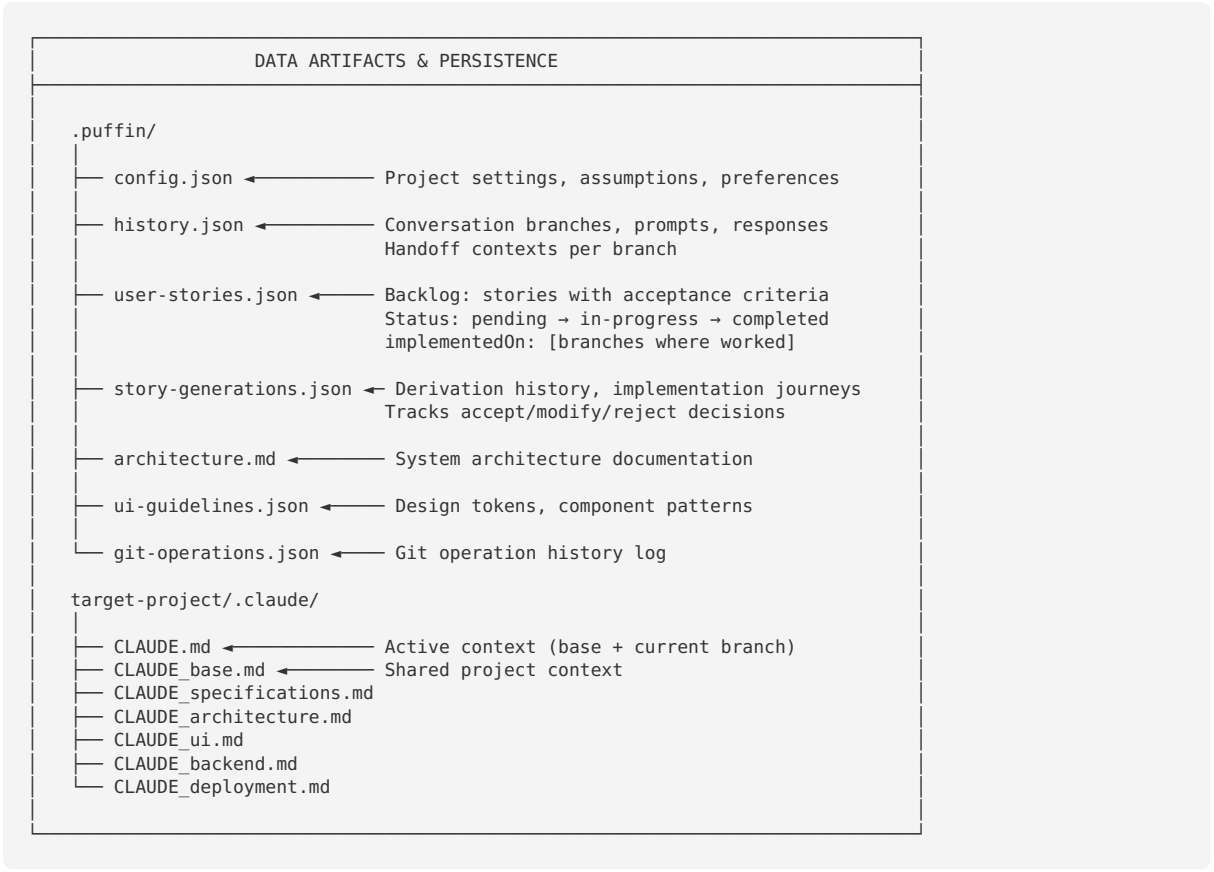
BRANCH TYPES & CONTEXT

Branch	Purpose	Context Provided to Claude
SPECIFICATIONS	Requirements & feature definitions	• Project description • Assumptions • User story template guidance • Acceptance criteria format
ARCHITECTURE	System design, APIs, data models	• Full architecture document • System boundaries • API contracts • Data model definitions
UI	Frontend implementation	• Design tokens (colors, spacing) • Component patterns (HTML/CSS) • Interaction guidelines • Typography scale
BACKEND	Server-side services	• Data model from architecture • API design hints • Authentication patterns • Database schema
DEPLOYMENT	DevOps & deployment	• Infrastructure notes • Deployment strategies • Environment configuration
TMP	Temporary experiments	• (No special context) • Scratch space

Complete End-to-End Flow



Information Artifacts & Persistence



This diagram captures all the processes Puffin supports from initial specification through to git push, including:

1. **Specification capture** on the specifications branch
2. **User story derivation** with review and editing
3. **Backlog management** with status tracking
4. **Sprint creation** with 4-story limit
5. **Planning phase** with Claude-generated plans
6. **Implementation** with branch-specific context
7. **Auto-continue** scoped to active story with acceptance criteria
8. **Stuck detection** and iteration limits
9. **Handoff system** for context continuity
10. **Git operations** (branch, stage, commit, push, merge)
11. **Dynamic CLAUDE.md** generation per branch