Sprint Summary: 2026-01-21_issue-13-pdf-export

Sprint ID: 2026-01-21_issue-13-pdf-export
Status: in-progress

Statistics

Started: 2026-01-21T14:49:30.136Z
Phases: 0/47
Steps: 0/5

Phases

worktree-setup [completed]

Create a git worktree for isolated sprint development.

Why Worktrees?
Git worktrees allow multiple sprints to run in parallel, each with its own:
- Working directory
- Branch
- Independent development state

Step 1: Check Current Location
```bash
# Determine if we're in a worktree or main repo
git rev-parse --show-toplevel
git worktree list
```

Step 2: Determine Worktree Location
The worktree structure should be:
```
project-root/
  trees/
    2026-01-21_issue-13-pdf-export/   # !• Sprint worktree (we'll create this)
  main/              # or project files directly here
```

If the project root doesn't have a `trees/` directory, create it:
```bash
mkdir -p trees
```

Step 3: Create the Worktree
```bash
# Create worktree with new branch
git worktree add trees/2026-01-21_issue-13-pdf-export -b sprint/2026-01-21_issue-13-pdf-export
```

If the branch already exists:
```bash
git worktree add trees/2026-01-21_issue-13-pdf-export sprint/2026-01-21_issue-13-pdf-export
```

Step 4: Verify Worktree
```bash
cd trees/2026-01-21_issue-13-pdf-export
```

git status
pwd
```

## Step 5: Copy Sprint Files to Worktree
The sprint definition needs to be accessible from the worktree:
```bash
# Copy the sprint directory to the worktree
cp -r .claude/sprints/2026-01-21_issue-13-pdf-export trees/2026-01-21_issue-13-pdf-export/.claude/sprints/
```

Step 6: Record Worktree Path
Create: context/worktree-info.md

```markdown
# Worktree Information

## Paths
- **Worktree**: trees/2026-01-21_issue-13-pdf-export
- **Branch**: sprint/2026-01-21_issue-13-pdf-export
- **Main repo**: [path to main repo]

## Working in the Worktree
All subsequent phases will work in the worktree directory.

## After Sprint Completion
- Create PR from sprint/2026-01-21_issue-13-pdf-export branch
- Clean up worktree: `git worktree remove trees/2026-01-21_issue-13-pdf-export`
```

Output
- Worktree created at `trees/2026-01-21_issue-13-pdf-export`
- Branch `sprint/2026-01-21_issue-13-pdf-export` created
- Sprint files copied to worktree
- Worktree info documented

IMPORTANT
All subsequent phases must work within the worktree:
`cd trees/2026-01-21_issue-13-pdf-export` before any operations.

preflight [completed]
Create comprehensive sprint context for TDD-based plugin development.

Your Task
Analyze the sprint scope and prepare shared context for all development phases.

Step 0: Switch to Worktree
```bash
# Ensure we're working in the sprint worktree
cd trees/2026-01-21_issue-13-pdf-export
pwd  # Verify location
```

Step 1: Verify Branch
```bash
git branch --show-current  # Should show: sprint/2026-01-21_issue-13-pdf-export
git status
```

Step 2: Analyze Sprint Scope
Read SPRINT.yaml to understand:
- All steps and their relationships
- Overall sprint goal
- Technical requirements
```

- Dependencies between steps

Step 3: Research Project Context
Investigate the codebase to understand:
- Project architecture and structure
- Existing patterns and conventions
- Test frameworks and patterns used
- Build/test/lint commands
- Documentation structure

Step 4: Documentation Inventory
Catalog existing documentation that may need updates:
```bash
# Find all documentation files
find . -name "*.md" -type f | grep -E "(README|docs/|GUIDE|REFERENCE)" | head -50
```

Create a documentation map showing what exists and may need updates.

Step 5: Generate Shared Context
Create: context/_shared-context.md

```markdown
# Shared Sprint Context

## Project Architecture
[High-level architecture overview relevant to this sprint]

## Test Patterns
- Test framework: [framework used]
- Test file location: [where tests live]
- Test naming: [naming conventions]
- Mocking patterns: [how mocks are done]

## Key Patterns
- [Pattern 1]: [Where and how it's used]
- [Pattern 2]: [Where and how it's used]

## Commands
- Build: `[build command]`
- Test: `[test command]`
- Test (watch): `[watch command if available]`
- Lint: `[lint command]`
- TypeCheck: `[typecheck command]`

## Documentation Structure
- User Guide: [path or "does not exist"]
- Getting Started: [path or "does not exist"]
- Reference: [path or "does not exist"]
- API Docs: [path or "does not exist"]

## Dependencies
### Internal Modules
- [Module]: [Purpose]

### External Packages
- [Package]: [Usage]
```

Step 6: Generate Sprint Plan
Create: context/sprint-plan.md

```markdown
# Sprint Plan: 2026-01-21_issue-13-pdf-export
```

Goal
[One paragraph describing what this sprint accomplishes]

TDD Approach
Each step follows: RED !’ GREEN !’ REFACTOR !’ QA

Success Criteria
- [ ] All gherkin scenarios pass (100% score)
- [ ] All unit tests pass
- [ ] Build passes
- [ ] Documentation updated

Step Breakdown

Step 0: [Step title]
Scope: [What this step does]
Tests to Write: [Key test cases]
Files: [Expected files to create/modify]
Docs Impact: [Documentation that may need updates]

[Continue for all steps]

Documentation Update Plan
Doc	Status	Updates Needed
User Guide	[exists/new]	[description]
Getting Started	[exists/new]	[description]
Reference	[exists/new]	[description]
```

## Output
- Sprint branch created
- context/_shared-context.md with project patterns
- context/sprint-plan.md with TDD plan
- Commit preflight artifacts:
  ```bash
  git add context/
  git commit -m "preflight: add shared context and TDD sprint plan"
  ```

development [in-progress]

step-0 [completed]

Step 1: Research and setup PDF generation infrastructure.

Tasks:
- Analyze existing sprint summary structure and data available
- Research PDF generation options (puppeteer, pdfkit, jspdf, etc.)
- Choose library that supports both text and chart/image embedding
- Add required dependencies to the plugin
- Create basic PDF generation utility module

Output: Working PDF generation setup with dependency installed.

context [completed]

Prepare context for TDD development.

Step 0: Ensure Working in Worktree (if applicable)
```bash
# If trees directory exists, switch to worktree
if [ -d "trees/2026-01-21_issue-13-pdf-export" ]; then
  cd trees/2026-01-21_issue-13-pdf-export
fi
pwd
git branch --show-current
```

Step 1: Read Shared Context
Read: context/_shared-context.md (project patterns, test patterns, commands)
```

Read: context/sprint-plan.md (how this step fits in the sprint)

Step 2: Understand the Step
Step 1: Research and setup PDF generation infrastructure.

Tasks:
- Analyze existing sprint summary structure and data available
- Research PDF generation options (puppeteer, pdfkit, jspdf, etc.)
- Choose library that supports both text and chart/image embedding
- Add required dependencies to the plugin
- Create basic PDF generation utility module

Output: Working PDF generation setup with dependency installed.

Output
Confirm understanding of:
- Test patterns and locations
- Build/test commands
- The step requirements

red [completed]
Write tests and gherkin scenarios BEFORE any implementation.

Your Task
Step 1: Research and setup PDF generation infrastructure.

Tasks:
- Analyze existing sprint summary structure and data available
- Research PDF generation options (puppeteer, pdfkit, jspdf, etc.)
- Choose library that supports both text and chart/image embedding
- Add required dependencies to the plugin
- Create basic PDF generation utility module

Output: Working PDF generation setup with dependency installed.

Shared Context
Read: context/_shared-context.md (project patterns, test patterns, commands)
Read: context/sprint-plan.md (how this step fits in the sprint)

TDD Rule: RED Phase
In this phase, you ONLY write tests. No implementation code.
Tests should FAIL when run (because implementation doesn't exist yet).

Instructions

Part 1: Write Gherkin Scenarios (4-8 scenarios)
Create scenarios that define the expected behavior:

```gherkin
Scenario: [Descriptive name]
  Given [precondition or context]
  When [action taken]
  Then [expected outcome]

Verification: `[shell command that exits 0 on success, non-zero on failure]`
Pass: Exit code = 0 !' Score 1
Fail: Exit code "` 0 !' Score 0
```

Part 2: Write Unit Tests
Create test files that exercise the expected functionality:

1. Determine test file location (from _shared-context.md patterns)
2. Write test cases that match gherkin scenarios
3. Include edge cases and error handling tests
4. Tests should be runnable but FAIL (code doesn't exist)

Test File Template
```typescript
// Filename follows project convention from _shared-context.md
import { describe, it, expect } from '[test-framework]';

describe('[Feature under test]', () => {
  describe('[Scenario group]', () => {
    it('should [expected behavior from gherkin]', () => {
      // Arrange - Given
      // Act - When
      // Assert - Then
    });
```

```
    it('should [another expected behavior]', () => {
      // ...
    });
  });

  describe('edge cases', () => {
    it('should handle [edge case]', () => {
      // ...
    });
  });

  describe('error handling', () => {
    it('should throw when [error condition]', () => {
      // ...
    });
  });
});
```

Output
Create: artifacts/step-0-gherkin.md

```markdown
# Gherkin Scenarios: step-0

## Step Task
Step 1: Research and setup PDF generation infrastructure.

Tasks:
- Analyze existing sprint summary structure and data available
- Research PDF generation options (puppeteer, pdfkit, jspdf, etc.)
- Choose library that supports both text and chart/image embedding
- Add required dependencies to the plugin
- Create basic PDF generation utility module

Output: Working PDF generation setup with dependency installed.


## Success Criteria
All scenarios must pass (score = 1) for the step to be complete.
Total scenarios: [N]
Required score: [N]/[N]

---

## Scenario 1: [Name]
[Full gherkin with verification]

---

## Scenario 2: [Name]
[Continue for 4-8 scenarios]

---

## Unit Test Coverage
| Test File | Test Cases | Scenarios Covered |
|-----------|------------|-------------------|
| [path] | [count] | 1, 2, 3 |

## RED Phase Verification
Tests are expected to FAIL at this point:
\`\`\`bash
npm test -- --testPathPattern="[pattern]"
# Expected: FAIL (no implementation yet)
\`\`\`
```

Commit (Tests Only)
```bash
git add [test files] artifacts/step-0-gherkin.md
git commit -m "test(step-0): add failing tests [RED]"
```

Important
- Do NOT write any implementation code in this phase
- Tests SHOULD fail - that's the point of RED
- Each gherkin scenario should have corresponding unit test(s)

green-context [completed]

Gather step-specific context for implementation.

Your Task
Step 1: Research and setup PDF generation infrastructure.

Tasks:
- Analyze existing sprint summary structure and data available
- Research PDF generation options (puppeteer, pdfkit, jspdf, etc.)
- Choose library that supports both text and chart/image embedding
- Add required dependencies to the plugin
- Create basic PDF generation utility module

Output: Working PDF generation setup with dependency installed.

Shared Context
Read: context/_shared-context.md (project-wide patterns)
Read: artifacts/step-0-gherkin.md (scenarios to implement)

Instructions
Research the specific code areas this step will touch:

1. **Related Existing Code**
   - Find similar implementations to follow as patterns
   - Identify modules to import from
   - Note interfaces/types to implement or extend

2. **Dependencies**
   - Internal module imports needed
   - External packages needed
   - Version constraints

3. **Integration Points**
   - How will this code be called/used?
   - What modules will import from this?
   - Existing tests to reference

4. **Patterns for Implementation**
   - Error handling patterns
   - Naming conventions
   - File organization

Output
Create: context/step-0-context.md

```markdown
# Step Context: step-0

## Task
Step 1: Research and setup PDF generation infrastructure.

Tasks:
- Analyze existing sprint summary structure and data available
- Research PDF generation options (puppeteer, pdfkit, jspdf, etc.)
- Choose library that supports both text and chart/image embedding
- Add required dependencies to the plugin
- Create basic PDF generation utility module

Output: Working PDF generation setup with dependency installed.

## Implementation Plan
Based on gherkin scenarios, implement in this order:
1. [First thing to implement]
2. [Second thing]
...

## Related Code Patterns

### Pattern from: [path]
\`\`\`typescript
// Key pattern to follow
[relevant code snippet]
\`\`\`

## Required Imports
### Internal
- `[module]`: [what to import]

### External
```

- `[package]`: [what to import]

Types/Interfaces to Use
\`\`\`typescript
// From [source]
interface [Name] {
  [relevant fields]
}
\`\`\`

Integration Points
- Called by: [consumers]
- Calls: [dependencies]

Files to Create/Modify
File	Action	Purpose
[path]	Create	[purpose]
[path]	Modify	[changes]
```

## Commit
```bash
git add context/step-0-context.md
git commit -m "context(step-0): gather implementation context"
```

green [completed]
Implement MINIMAL code to make tests pass.

Your Task
Step 1: Research and setup PDF generation infrastructure.

Tasks:
- Analyze existing sprint summary structure and data available
- Research PDF generation options (puppeteer, pdfkit, jspdf, etc.)
- Choose library that supports both text and chart/image embedding
- Add required dependencies to the plugin
- Create basic PDF generation utility module

Output: Working PDF generation setup with dependency installed.

TDD Rule: GREEN Phase
Write the simplest code that makes tests pass. No more, no less.
"Make it work" - not beautiful, just working.

MUST READ BEFORE IMPLEMENTING
Read in order:
1. context/_shared-context.md (project patterns)
2. artifacts/step-0-gherkin.md (scenarios to satisfy)
3. context/step-0-context.md (specific patterns, imports)

Implementation Rules

Follow the Tests
Your implementation MUST satisfy:
- ALL gherkin scenario verifications
- ALL unit tests written in RED phase

Minimal Implementation
- Write the simplest code that passes tests
- Don't add features not covered by tests
- Don't optimize prematurely
- Don't add "nice to have" abstractions

Follow the Context
- Use patterns from context file
- Import from specified modules
- Follow naming conventions

Run Tests Frequently
```bash
# Run tests as you implement
npm test -- --testPathPattern="[pattern]" --watch
```

Commits
Make atomic commits for logical units:
```bash
git add [files]

```
git commit -m "feat(step-0): [what now works] [GREEN]"
```

Success Criteria
Before marking complete:
- [ ] All unit tests pass
- [ ] All gherkin verifications should pass
- [ ] No TypeScript errors
- [ ] Code follows context patterns

Run Final Test
```bash
npm test -- --testPathPattern="[pattern]"
# Expected: PASS (all tests green)
```

Important
- Tests MUST pass before moving to REFACTOR
- Don't refactor yet - that's the next phase
- "Quick and dirty" is OK - we'll clean up next

refactor [completed]

Refactor implementation while keeping tests green.

Your Task
Step 1: Research and setup PDF generation infrastructure.

Tasks:
- Analyze existing sprint summary structure and data available
- Research PDF generation options (puppeteer, pdfkit, jspdf, etc.)
- Choose library that supports both text and chart/image embedding
- Add required dependencies to the plugin
- Create basic PDF generation utility module

Output: Working PDF generation setup with dependency installed.

TDD Rule: REFACTOR Phase
"Make it right" - improve code quality without changing behavior.
Tests MUST stay green throughout refactoring.

Context
Read: context/step-0-context.md (target patterns)
Read: artifacts/step-0-gherkin.md (scenarios that must stay passing)

Refactoring Checklist

Code Quality
- [ ] Remove any duplication
- [ ] Improve naming (variables, functions, files)
- [ ] Extract helper functions if beneficial
- [ ] Simplify complex conditionals
- [ ] Add necessary type annotations

Pattern Conformance
- [ ] Follows project conventions from context
- [ ] Error handling matches patterns
- [ ] File organization matches patterns

Clean Code
- [ ] Remove dead code
- [ ] Remove console.log/debug statements
- [ ] Fix any linting issues
- [ ] Add JSDoc for public APIs only

Refactoring Rules

Run Tests After Each Change
```bash
npm test -- --testPathPattern="[pattern]"
```
If tests fail, REVERT the change immediately.

Small Steps
- One refactoring at a time
- Commit after each successful refactoring
- Don't refactor multiple things at once

Don't Change Behavior
- No new features
- No API changes (unless fixing test file imports)

- Same inputs !' same outputs

Commits
```bash
git add [files]
git commit -m "refactor(step-0): [improvement made]"
```

Lint and Type Check
```bash
npm run lint -- [files]
npm run typecheck
```

Fix any issues found.

Final Verification
```bash
npm test -- --testPathPattern="[pattern]"
npm run lint
npm run typecheck
```

All must pass before completing this phase.

qa [completed]

Verify implementation against gherkin scenarios.

Context
Read: artifacts/step-0-gherkin.md (scenarios with verification commands)

Step 1: Run Unit Tests
```bash
npm test -- --testPathPattern="[pattern]"
```

Record results: total tests, passed, failed.

Step 2: Run Each Gherkin Verification

For EACH scenario in the gherkin file:
1. Run the verification command
2. Record result: 1 (pass) or 0 (fail)
3. If fail, capture error output

Step 3: Calculate Score

Score = (passed scenarios) / (total scenarios)
- Score = 1.0 (all pass) !' PASS
- Score < 1.0 (any fail) !' FAIL

Step 4: Generate QA Report

Create: artifacts/step-0-qa-report.md

```markdown
# QA Report: step-0

## Summary
- Gherkin Scenarios: [N] total, [N] passed, [N] failed
- Gherkin Score: [X]/[N] = [percentage]%
- Unit Tests: [N] total, [N] passed, [N] failed

## Unit Test Results
\`\`\`
[test output]
\`\`\`

## Gherkin Verification Results

| # | Scenario | Result | Details |
|---|----------|--------|---------|
| 1 | [Name] | PASS/FAIL | [output] |
| 2 | [Name] | PASS/FAIL | [output] |
...

## Detailed Results

### Scenario 1: [Name]
**Verification**: `[command]`
**Exit Code**: [0 or N]
```

Output:
\`\`\`
[actual output]
\`\`\`
Result: PASS / FAIL

[Continue for all scenarios]

TDD Cycle Summary
Phase	Status
RED (tests)	' Completed
GREEN (implement)	' Completed
REFACTOR	' Completed
QA (verify)	PASS/FAIL

Issues Found
[If any failures, describe each issue]

Status: PASS / FAIL
```

## Step 5: Handle Outcome

### If ALL pass (Score = 100%):
```bash
git add artifacts/step-0-qa-report.md
git commit -m "qa(step-0): all scenarios passed"
```

If ANY fail (Score < 100%):
Inject fix phases:

```bash
PROGRESS_FILE=$(find .claude/sprints -name "PROGRESS.yaml" -type f 2>/dev/null | head -1)

yq -i '
  (.phases[] | select(.steps) | .steps[] | select(.status == "in-progress") | .phases) +=
  [
    {
      "id": "fix",
      "status": "pending",
      "prompt": "Fix failing scenarios for step-0.\n\nRead: artifacts/step-0-qa-report.md\n\n1. Review each FAIL scenario\n2. Fix implementation to pass\n3. Run verification to confirm\n4. Run unit tests to ensure no regression\n5. Commit fixes"
    },
    {
      "id": "reverify",
      "status": "pending",
      "prompt": "Re-verify all scenarios after fixes.\n\nRun ALL verifications from artifacts/step-0-gherkin.md\nRun ALL unit tests.\n\nIf ALL pass: Update QA report with PASS status.\nIf ANY fail: Inject another fix phase."
    }
  ]
' "$PROGRESS_FILE"

yq -i '
  (.. | select(.id == "qa" and .status == "in-progress")) |=
  (.status = "failed" | .error = "QA failed - fix phase injected")
' "$PROGRESS_FILE"
```

verify [completed]
Final integration verification for the step.

Your Task
Step 1: Research and setup PDF generation infrastructure.

Tasks:
- Analyze existing sprint summary structure and data available
- Research PDF generation options (puppeteer, pdfkit, jspdf, etc.)
- Choose library that supports both text and chart/image embedding
- Add required dependencies to the plugin
- Create basic PDF generation utility module

Output: Working PDF generation setup with dependency installed.

Context
Read: context/_shared-context.md (build commands)
Read: artifacts/step-0-qa-report.md (should show PASS)

Step 1: Full Test Suite
```

Run all tests (not just this step's tests):
```bash
npm test
```

Verify no regressions in other tests.

Step 2: Build Verification
```bash
npm run build
npm run typecheck
npm run lint
```

Step 3: Integration Check
- Check imports resolve correctly
- Verify no circular dependencies
- Ensure types are compatible

Step 4: Final Commit
```bash
git add -A
git status
git diff --cached --quiet || git commit -m "verify(step-0): integration verified"
```

Step Complete
The TDD cycle for this step is complete:
- ' RED: Tests written first
- ' GREEN: Minimal implementation
- ' REFACTOR: Code cleaned up
- ' QA: All scenarios pass
- ' VERIFY: Integration confirmed

step-1 [completed]

Step 2: Implement core PDF export with sprint data.

Tasks:
- Create PDF template/layout for sprint summary
- Include sprint metadata (name, dates, status)
- Add step listing with status indicators
- Include timing information and completion percentages
- Format text content for readability (headers, sections, spacing)

Output: Basic PDF export that renders sprint text content.

context [completed]

Prepare context for TDD development.

Step 0: Ensure Working in Worktree (if applicable)
```bash
# If trees directory exists, switch to worktree
if [ -d "trees/2026-01-21_issue-13-pdf-export" ]; then
  cd trees/2026-01-21_issue-13-pdf-export
fi
pwd
git branch --show-current
```

Step 1: Read Shared Context
Read: context/_shared-context.md (project patterns, test patterns, commands)
Read: context/sprint-plan.md (how this step fits in the sprint)

Step 2: Understand the Step
Step 2: Implement core PDF export with sprint data.

Tasks:
- Create PDF template/layout for sprint summary
- Include sprint metadata (name, dates, status)
- Add step listing with status indicators
- Include timing information and completion percentages
- Format text content for readability (headers, sections, spacing)

Output: Basic PDF export that renders sprint text content.

Output
Confirm understanding of:
- Test patterns and locations
- Build/test commands

- The step requirements
red [completed]
Write tests and gherkin scenarios BEFORE any implementation.

Your Task
Step 2: Implement core PDF export with sprint data.

Tasks:
- Create PDF template/layout for sprint summary
- Include sprint metadata (name, dates, status)
- Add step listing with status indicators
- Include timing information and completion percentages
- Format text content for readability (headers, sections, spacing)

Output: Basic PDF export that renders sprint text content.

Shared Context
Read: context/_shared-context.md (project patterns, test patterns, commands)
Read: context/sprint-plan.md (how this step fits in the sprint)

TDD Rule: RED Phase
In this phase, you ONLY write tests. No implementation code.
Tests should FAIL when run (because implementation doesn't exist yet).

Instructions

Part 1: Write Gherkin Scenarios (4-8 scenarios)
Create scenarios that define the expected behavior:

```gherkin
Scenario: [Descriptive name]
  Given [precondition or context]
  When [action taken]
  Then [expected outcome]

Verification: `[shell command that exits 0 on success, non-zero on failure]`
Pass: Exit code = 0 !' Score 1
Fail: Exit code "` 0 !' Score 0
```

Part 2: Write Unit Tests
Create test files that exercise the expected functionality:

1. Determine test file location (from _shared-context.md patterns)
2. Write test cases that match gherkin scenarios
3. Include edge cases and error handling tests
4. Tests should be runnable but FAIL (code doesn't exist)

Test File Template
```typescript
// Filename follows project convention from _shared-context.md
import { describe, it, expect } from '[test-framework]';

describe('[Feature under test]', () => {
  describe('[Scenario group]', () => {
    it('should [expected behavior from gherkin]', () => {
      // Arrange - Given
      // Act - When
      // Assert - Then
    });

    it('should [another expected behavior]', () => {
      // ...
    });
  });

  describe('edge cases', () => {
    it('should handle [edge case]', () => {
      // ...
    });
  });

  describe('error handling', () => {
    it('should throw when [error condition]', () => {
      // ...
    });
  });
});
```

Output
Create: artifacts/step-1-gherkin.md

```markdown
# Gherkin Scenarios: step-1

## Step Task
Step 2: Implement core PDF export with sprint data.

Tasks:
- Create PDF template/layout for sprint summary
- Include sprint metadata (name, dates, status)
- Add step listing with status indicators
- Include timing information and completion percentages
- Format text content for readability (headers, sections, spacing)

Output: Basic PDF export that renders sprint text content.


## Success Criteria
All scenarios must pass (score = 1) for the step to be complete.
Total scenarios: [N]
Required score: [N]/[N]

---

## Scenario 1: [Name]
[Full gherkin with verification]

---

## Scenario 2: [Name]
[Continue for 4-8 scenarios]

---

## Unit Test Coverage
| Test File | Test Cases | Scenarios Covered |
|----------|-----------|-------------------|
| [path] | [count] | 1, 2, 3 |

## RED Phase Verification
Tests are expected to FAIL at this point:
\`\`\`bash
npm test -- --testPathPattern="[pattern]"
# Expected: FAIL (no implementation yet)
\`\`\`
```

Commit (Tests Only)
```bash
git add [test files] artifacts/step-1-gherkin.md
git commit -m "test(step-1): add failing tests [RED]"
```

Important
- Do NOT write any implementation code in this phase
- Tests SHOULD fail - that's the point of RED
- Each gherkin scenario should have corresponding unit test(s)

green-context [completed]

Gather step-specific context for implementation.

Your Task
Step 2: Implement core PDF export with sprint data.

Tasks:
- Create PDF template/layout for sprint summary
- Include sprint metadata (name, dates, status)
- Add step listing with status indicators
- Include timing information and completion percentages
- Format text content for readability (headers, sections, spacing)

Output: Basic PDF export that renders sprint text content.

Shared Context
Read: context/_shared-context.md (project-wide patterns)
Read: artifacts/step-1-gherkin.md (scenarios to implement)

Instructions
Research the specific code areas this step will touch:

1. **Related Existing Code**
   - Find similar implementations to follow as patterns
   - Identify modules to import from
   - Note interfaces/types to implement or extend

2. **Dependencies**
   - Internal module imports needed
   - External packages needed
   - Version constraints

3. **Integration Points**
   - How will this code be called/used?
   - What modules will import from this?
   - Existing tests to reference

4. **Patterns for Implementation**
   - Error handling patterns
   - Naming conventions
   - File organization

Output
Create: context/step-1-context.md

```markdown
# Step Context: step-1

## Task
Step 2: Implement core PDF export with sprint data.

Tasks:
- Create PDF template/layout for sprint summary
- Include sprint metadata (name, dates, status)
- Add step listing with status indicators
- Include timing information and completion percentages
- Format text content for readability (headers, sections, spacing)

Output: Basic PDF export that renders sprint text content.


## Implementation Plan
Based on gherkin scenarios, implement in this order:
1. [First thing to implement]
2. [Second thing]
...

## Related Code Patterns

### Pattern from: [path]
\`\`\`typescript
// Key pattern to follow
[relevant code snippet]
\`\`\`

## Required Imports
### Internal
- `[module]`: [what to import]

### External
- `[package]`: [what to import]

## Types/Interfaces to Use
\`\`\`typescript
// From [source]
interface [Name] {
  [relevant fields]
}
\`\`\`

## Integration Points
- Called by: [consumers]
- Calls: [dependencies]

## Files to Create/Modify
| File | Action | Purpose |
|------|--------|---------|
| [path] | Create | [purpose] |
| [path] | Modify | [changes] |
```

```

## Commit
```bash
git add context/step-1-context.md
git commit -m "context(step-1): gather implementation context"
```
green [completed]
Implement MINIMAL code to make tests pass.

Your Task
Step 2: Implement core PDF export with sprint data.

Tasks:
- Create PDF template/layout for sprint summary
- Include sprint metadata (name, dates, status)
- Add step listing with status indicators
- Include timing information and completion percentages
- Format text content for readability (headers, sections, spacing)

Output: Basic PDF export that renders sprint text content.

TDD Rule: GREEN Phase
Write the simplest code that makes tests pass. No more, no less.
"Make it work" - not beautiful, just working.

MUST READ BEFORE IMPLEMENTING
Read in order:
1. context/_shared-context.md (project patterns)
2. artifacts/step-1-gherkin.md (scenarios to satisfy)
3. context/step-1-context.md (specific patterns, imports)

Implementation Rules

Follow the Tests
Your implementation MUST satisfy:
- ALL gherkin scenario verifications
- ALL unit tests written in RED phase

Minimal Implementation
- Write the simplest code that passes tests
- Don't add features not covered by tests
- Don't optimize prematurely
- Don't add "nice to have" abstractions

Follow the Context
- Use patterns from context file
- Import from specified modules
- Follow naming conventions

Run Tests Frequently
```bash
# Run tests as you implement
npm test -- --testPathPattern="[pattern]" --watch
```

Commits
Make atomic commits for logical units:
```bash
git add [files]
git commit -m "feat(step-1): [what now works] [GREEN]"
```

Success Criteria
Before marking complete:
- [ ] All unit tests pass
- [ ] All gherkin verifications should pass
- [ ] No TypeScript errors
- [ ] Code follows context patterns

Run Final Test
```bash
npm test -- --testPathPattern="[pattern]"
# Expected: PASS (all tests green)
```

Important
- Tests MUST pass before moving to REFACTOR
- Don't refactor yet - that's the next phase
```

- "Quick and dirty" is OK - we'll clean up next

refactor [completed]

Refactor implementation while keeping tests green.

Your Task
Step 2: Implement core PDF export with sprint data.

Tasks:
- Create PDF template/layout for sprint summary
- Include sprint metadata (name, dates, status)
- Add step listing with status indicators
- Include timing information and completion percentages
- Format text content for readability (headers, sections, spacing)

Output: Basic PDF export that renders sprint text content.

TDD Rule: REFACTOR Phase
"Make it right" - improve code quality without changing behavior.
Tests MUST stay green throughout refactoring.

Context
Read: context/step-1-context.md (target patterns)
Read: artifacts/step-1-gherkin.md (scenarios that must stay passing)

Refactoring Checklist

Code Quality
- [ ] Remove any duplication
- [ ] Improve naming (variables, functions, files)
- [ ] Extract helper functions if beneficial
- [ ] Simplify complex conditionals
- [ ] Add necessary type annotations

Pattern Conformance
- [ ] Follows project conventions from context
- [ ] Error handling matches patterns
- [ ] File organization matches patterns

Clean Code
- [ ] Remove dead code
- [ ] Remove console.log/debug statements
- [ ] Fix any linting issues
- [ ] Add JSDoc for public APIs only

Refactoring Rules

Run Tests After Each Change
```bash
npm test -- --testPathPattern="[pattern]"
```
If tests fail, REVERT the change immediately.

Small Steps
- One refactoring at a time
- Commit after each successful refactoring
- Don't refactor multiple things at once

Don't Change Behavior
- No new features
- No API changes (unless fixing test file imports)
- Same inputs !' same outputs

Commits
```bash
git add [files]
git commit -m "refactor(step-1): [improvement made]"
```

Lint and Type Check
```bash
npm run lint -- [files]
npm run typecheck
```

Fix any issues found.

Final Verification
```bash
npm test -- --testPathPattern="[pattern]"
```

```
npm run lint
npm run typecheck
```

All must pass before completing this phase.

qa [completed]

Verify implementation against gherkin scenarios.

Context
Read: artifacts/step-1-gherkin.md (scenarios with verification commands)

Step 1: Run Unit Tests
```bash
npm test -- --testPathPattern="[pattern]"
```

Record results: total tests, passed, failed.

Step 2: Run Each Gherkin Verification

For EACH scenario in the gherkin file:
1. Run the verification command
2. Record result: 1 (pass) or 0 (fail)
3. If fail, capture error output

Step 3: Calculate Score

Score = (passed scenarios) / (total scenarios)
- Score = 1.0 (all pass) !' PASS
- Score < 1.0 (any fail) !' FAIL

Step 4: Generate QA Report

Create: artifacts/step-1-qa-report.md

```markdown
# QA Report: step-1

## Summary
- Gherkin Scenarios: [N] total, [N] passed, [N] failed
- Gherkin Score: [X]/[N] = [percentage]%
- Unit Tests: [N] total, [N] passed, [N] failed

## Unit Test Results
\`\`\`
[test output]
\`\`\`

## Gherkin Verification Results

| # | Scenario | Result | Details |
|---|----------|--------|---------|
| 1 | [Name] | PASS/FAIL | [output] |
| 2 | [Name] | PASS/FAIL | [output] |
...

## Detailed Results

### Scenario 1: [Name]
**Verification**: `[command]`
**Exit Code**: [0 or N]
**Output**:
\`\`\`
[actual output]
\`\`\`
**Result**: PASS / FAIL

[Continue for all scenarios]

## TDD Cycle Summary
| Phase | Status |
|-------|--------|
| RED (tests) | '  Completed |
| GREEN (implement) | '  Completed |
| REFACTOR | '  Completed |
| QA (verify) | PASS/FAIL |

## Issues Found
[If any failures, describe each issue]
```

Status: PASS / FAIL
```

## Step 5: Handle Outcome

### If ALL pass (Score = 100%):
```bash
git add artifacts/step-1-qa-report.md
git commit -m "qa(step-1): all scenarios passed"
```

If ANY fail (Score < 100%):
Inject fix phases:

```bash
PROGRESS_FILE=$(find .claude/sprints -name "PROGRESS.yaml" -type f 2>/dev/null | head -1)

yq -i '
  (.phases[] | select(.steps) | .steps[] | select(.status == "in-progress") | .phases) +=
  [
    {
      "id": "fix",
      "status": "pending",
      "prompt": "Fix failing scenarios for step-1.\n\nRead: artifacts/step-1-qa-report.md\n\n1. Review each FAIL scenario\n2. Fix implementation to pass\n3. Run verification to confirm\n4. Run unit tests to ensure no regression\n5. Commit fixes"
    },
    {
      "id": "reverify",
      "status": "pending",
      "prompt": "Re-verify all scenarios after fixes.\n\nRun ALL verifications from artifacts/step-1-gherkin.md\nRun ALL unit tests.\n\nIf ALL pass: Update QA report with PASS status.\nIf ANY fail: Inject another fix phase."
    }
  ]
' "$PROGRESS_FILE"

yq -i '
  (.. | select(.id == "qa" and .status == "in-progress")) |=
  (.status = "failed" | .error = "QA failed - fix phase injected")
' "$PROGRESS_FILE"
```

verify [completed]
Final integration verification for the step.

Your Task
Step 2: Implement core PDF export with sprint data.

Tasks:
- Create PDF template/layout for sprint summary
- Include sprint metadata (name, dates, status)
- Add step listing with status indicators
- Include timing information and completion percentages
- Format text content for readability (headers, sections, spacing)

Output: Basic PDF export that renders sprint text content.

Context
Read: context/_shared-context.md (build commands)
Read: artifacts/step-1-qa-report.md (should show PASS)

Step 1: Full Test Suite
Run all tests (not just this step's tests):
```bash
npm test
```

Verify no regressions in other tests.

Step 2: Build Verification
```bash
npm run build
npm run typecheck
npm run lint
```

Step 3: Integration Check
- Check imports resolve correctly
- Verify no circular dependencies
- Ensure types are compatible
```

Step 4: Final Commit
```bash
git add -A
git status
git diff --cached --quiet || git commit -m "verify(step-1): integration verified"
```

Step Complete
The TDD cycle for this step is complete:
- ' RED: Tests written first
- ' GREEN: Minimal implementation
- ' REFACTOR: Code cleaned up
- ' QA: All scenarios pass
- ' VERIFY: Integration confirmed

step-2 [completed]

Step 3: Add visual progress chart to PDF.

Tasks:
- Generate visual chart showing sprint progress (e.g., pie chart, progress bar, or timeline)
- Chart should display: completed vs pending vs failed steps
- Consider adding timeline visualization if timestamps available
- Ensure chart renders cleanly in PDF format
- Use charting library compatible with PDF generation (e.g., chart.js with canvas, SVG-based)

Output: PDF now includes visual progress chart.

context [completed]

Prepare context for TDD development.

Step 0: Ensure Working in Worktree (if applicable)
```bash
# If trees directory exists, switch to worktree
if [ -d "trees/2026-01-21_issue-13-pdf-export" ]; then
  cd trees/2026-01-21_issue-13-pdf-export
fi
pwd
git branch --show-current
```

Step 1: Read Shared Context
Read: context/_shared-context.md (project patterns, test patterns, commands)
Read: context/sprint-plan.md (how this step fits in the sprint)

Step 2: Understand the Step
Step 3: Add visual progress chart to PDF.

Tasks:
- Generate visual chart showing sprint progress (e.g., pie chart, progress bar, or timeline)
- Chart should display: completed vs pending vs failed steps
- Consider adding timeline visualization if timestamps available
- Ensure chart renders cleanly in PDF format
- Use charting library compatible with PDF generation (e.g., chart.js with canvas, SVG-based)

Output: PDF now includes visual progress chart.

Output
Confirm understanding of:
- Test patterns and locations
- Build/test commands
- The step requirements

red [completed]

Write tests and gherkin scenarios BEFORE any implementation.

Your Task
Step 3: Add visual progress chart to PDF.

Tasks:
- Generate visual chart showing sprint progress (e.g., pie chart, progress bar, or timeline)
- Chart should display: completed vs pending vs failed steps
- Consider adding timeline visualization if timestamps available
- Ensure chart renders cleanly in PDF format
- Use charting library compatible with PDF generation (e.g., chart.js with canvas, SVG-based)

Output: PDF now includes visual progress chart.

Shared Context
Read: context/_shared-context.md (project patterns, test patterns, commands)

Read: context/sprint-plan.md (how this step fits in the sprint)

TDD Rule: RED Phase
In this phase, you ONLY write tests. No implementation code.
Tests should FAIL when run (because implementation doesn't exist yet).

Instructions

Part 1: Write Gherkin Scenarios (4-8 scenarios)
Create scenarios that define the expected behavior:

```gherkin
Scenario: [Descriptive name]
  Given [precondition or context]
  When [action taken]
  Then [expected outcome]

Verification: `[shell command that exits 0 on success, non-zero on failure]`
Pass: Exit code = 0 !' Score 1
Fail: Exit code "` 0 !' Score 0
```

Part 2: Write Unit Tests
Create test files that exercise the expected functionality:

1. Determine test file location (from _shared-context.md patterns)
2. Write test cases that match gherkin scenarios
3. Include edge cases and error handling tests
4. Tests should be runnable but FAIL (code doesn't exist)

Test File Template
```typescript
// Filename follows project convention from _shared-context.md
import { describe, it, expect } from '[test-framework]';

describe('[Feature under test]', () => {
  describe('[Scenario group]', () => {
    it('should [expected behavior from gherkin]', () => {
      // Arrange - Given
      // Act - When
      // Assert - Then
    });

    it('should [another expected behavior]', () => {
      // ...
    });
  });

  describe('edge cases', () => {
    it('should handle [edge case]', () => {
      // ...
    });
  });

  describe('error handling', () => {
    it('should throw when [error condition]', () => {
      // ...
    });
  });
});
```

Output
Create: artifacts/step-2-gherkin.md

```markdown
# Gherkin Scenarios: step-2

## Step Task
Step 3: Add visual progress chart to PDF.

Tasks:
- Generate visual chart showing sprint progress (e.g., pie chart, progress bar, or timeline)
- Chart should display: completed vs pending vs failed steps
- Consider adding timeline visualization if timestamps available
- Ensure chart renders cleanly in PDF format
- Use charting library compatible with PDF generation (e.g., chart.js with canvas, SVG-based)

Output: PDF now includes visual progress chart.
```

Success Criteria
All scenarios must pass (score = 1) for the step to be complete.
Total scenarios: [N]
Required score: [N]/[N]

Scenario 1: [Name]
[Full gherkin with verification]

Scenario 2: [Name]
[Continue for 4-8 scenarios]

Unit Test Coverage
Test File	Test Cases	Scenarios Covered
[path]	[count]	1, 2, 3

RED Phase Verification
Tests are expected to FAIL at this point:
\`\`\`bash
npm test -- --testPathPattern="[pattern]"
Expected: FAIL (no implementation yet)
\`\`\`
```

## Commit (Tests Only)
```bash
git add [test files] artifacts/step-2-gherkin.md
git commit -m "test(step-2): add failing tests [RED]"
```

Important
- Do NOT write any implementation code in this phase
- Tests SHOULD fail - that's the point of RED
- Each gherkin scenario should have corresponding unit test(s)

green-context [completed]

Gather step-specific context for implementation.

Your Task
Step 3: Add visual progress chart to PDF.

Tasks:
- Generate visual chart showing sprint progress (e.g., pie chart, progress bar, or timeline)
- Chart should display: completed vs pending vs failed steps
- Consider adding timeline visualization if timestamps available
- Ensure chart renders cleanly in PDF format
- Use charting library compatible with PDF generation (e.g., chart.js with canvas, SVG-based)

Output: PDF now includes visual progress chart.

Shared Context
Read: context/_shared-context.md (project-wide patterns)
Read: artifacts/step-2-gherkin.md (scenarios to implement)

Instructions
Research the specific code areas this step will touch:

1. **Related Existing Code**
   - Find similar implementations to follow as patterns
   - Identify modules to import from
   - Note interfaces/types to implement or extend

2. **Dependencies**
   - Internal module imports needed
   - External packages needed
   - Version constraints

3. **Integration Points**
   - How will this code be called/used?
   - What modules will import from this?
   - Existing tests to reference

4. **Patterns for Implementation**
```

- Error handling patterns
- Naming conventions
- File organization

Output
Create: context/step-2-context.md

```markdown
# Step Context: step-2

## Task
Step 3: Add visual progress chart to PDF.

Tasks:
- Generate visual chart showing sprint progress (e.g., pie chart, progress bar, or timeline)
- Chart should display: completed vs pending vs failed steps
- Consider adding timeline visualization if timestamps available
- Ensure chart renders cleanly in PDF format
- Use charting library compatible with PDF generation (e.g., chart.js with canvas, SVG-based)

Output: PDF now includes visual progress chart.


## Implementation Plan
Based on gherkin scenarios, implement in this order:
1. [First thing to implement]
2. [Second thing]
...

## Related Code Patterns

### Pattern from: [path]
\`\`\`typescript
// Key pattern to follow
[relevant code snippet]
\`\`\`

## Required Imports
### Internal
- `[module]`: [what to import]

### External
- `[package]`: [what to import]

## Types/Interfaces to Use
\`\`\`typescript
// From [source]
interface [Name] {
  [relevant fields]
}
\`\`\`

## Integration Points
- Called by: [consumers]
- Calls: [dependencies]

## Files to Create/Modify
| File | Action | Purpose |
|------|--------|---------|
| [path] | Create | [purpose] |
| [path] | Modify | [changes] |
```

Commit
```bash
git add context/step-2-context.md
git commit -m "context(step-2): gather implementation context"
```
green [completed]
Implement MINIMAL code to make tests pass.

Your Task
Step 3: Add visual progress chart to PDF.

Tasks:
- Generate visual chart showing sprint progress (e.g., pie chart, progress bar, or timeline)
- Chart should display: completed vs pending vs failed steps
- Consider adding timeline visualization if timestamps available
- Ensure chart renders cleanly in PDF format
- Use charting library compatible with PDF generation (e.g., chart.js with canvas, SVG-based)

Output: PDF now includes visual progress chart.

TDD Rule: GREEN Phase
Write the simplest code that makes tests pass. No more, no less.
"Make it work" - not beautiful, just working.

MUST READ BEFORE IMPLEMENTING
Read in order:
1. context/_shared-context.md (project patterns)
2. artifacts/step-2-gherkin.md (scenarios to satisfy)
3. context/step-2-context.md (specific patterns, imports)

Implementation Rules

Follow the Tests
Your implementation MUST satisfy:
- ALL gherkin scenario verifications
- ALL unit tests written in RED phase

Minimal Implementation
- Write the simplest code that passes tests
- Don't add features not covered by tests
- Don't optimize prematurely
- Don't add "nice to have" abstractions

Follow the Context
- Use patterns from context file
- Import from specified modules
- Follow naming conventions

Run Tests Frequently
```bash
# Run tests as you implement
npm test -- --testPathPattern="[pattern]" --watch
```

Commits
Make atomic commits for logical units:
```bash
git add [files]
git commit -m "feat(step-2): [what now works] [GREEN]"
```

Success Criteria
Before marking complete:
- [ ] All unit tests pass
- [ ] All gherkin verifications should pass
- [ ] No TypeScript errors
- [ ] Code follows context patterns

Run Final Test
```bash
npm test -- --testPathPattern="[pattern]"
# Expected: PASS (all tests green)
```

Important
- Tests MUST pass before moving to REFACTOR
- Don't refactor yet - that's the next phase
- "Quick and dirty" is OK - we'll clean up next
refactor [completed]
Refactor implementation while keeping tests green.

Your Task
Step 3: Add visual progress chart to PDF.

Tasks:
- Generate visual chart showing sprint progress (e.g., pie chart, progress bar, or timeline)
- Chart should display: completed vs pending vs failed steps
- Consider adding timeline visualization if timestamps available
- Ensure chart renders cleanly in PDF format
- Use charting library compatible with PDF generation (e.g., chart.js with canvas, SVG-based)

Output: PDF now includes visual progress chart.

TDD Rule: REFACTOR Phase
"Make it right" - improve code quality without changing behavior.

Tests MUST stay green throughout refactoring.

Context
Read: context/step-2-context.md (target patterns)
Read: artifacts/step-2-gherkin.md (scenarios that must stay passing)

Refactoring Checklist

Code Quality
- [ ] Remove any duplication
- [ ] Improve naming (variables, functions, files)
- [ ] Extract helper functions if beneficial
- [ ] Simplify complex conditionals
- [ ] Add necessary type annotations

Pattern Conformance
- [ ] Follows project conventions from context
- [ ] Error handling matches patterns
- [ ] File organization matches patterns

Clean Code
- [ ] Remove dead code
- [ ] Remove console.log/debug statements
- [ ] Fix any linting issues
- [ ] Add JSDoc for public APIs only

Refactoring Rules

Run Tests After Each Change
```bash
npm test -- --testPathPattern="[pattern]"
```
If tests fail, REVERT the change immediately.

Small Steps
- One refactoring at a time
- Commit after each successful refactoring
- Don't refactor multiple things at once

Don't Change Behavior
- No new features
- No API changes (unless fixing test file imports)
- Same inputs !' same outputs

Commits
```bash
git add [files]
git commit -m "refactor(step-2): [improvement made]"
```

Lint and Type Check
```bash
npm run lint -- [files]
npm run typecheck
```

Fix any issues found.

Final Verification
```bash
npm test -- --testPathPattern="[pattern]"
npm run lint
npm run typecheck
```

All must pass before completing this phase.
qa [completed]
Verify implementation against gherkin scenarios.

Context
Read: artifacts/step-2-gherkin.md (scenarios with verification commands)

Step 1: Run Unit Tests
```bash
npm test -- --testPathPattern="[pattern]"
```

Record results: total tests, passed, failed.

Step 2: Run Each Gherkin Verification

For EACH scenario in the gherkin file:
1. Run the verification command
2. Record result: 1 (pass) or 0 (fail)
3. If fail, capture error output

Step 3: Calculate Score

Score = (passed scenarios) / (total scenarios)
- Score = 1.0 (all pass) !' PASS
- Score < 1.0 (any fail) !' FAIL

Step 4: Generate QA Report

Create: artifacts/step-2-qa-report.md

```markdown
# QA Report: step-2

## Summary
- Gherkin Scenarios: [N] total, [N] passed, [N] failed
- Gherkin Score: [X]/[N] = [percentage]%
- Unit Tests: [N] total, [N] passed, [N] failed

## Unit Test Results
\`\`\`
[test output]
\`\`\`

## Gherkin Verification Results

| # | Scenario | Result | Details |
|---|----------|--------|---------|
| 1 | [Name] | PASS/FAIL | [output] |
| 2 | [Name] | PASS/FAIL | [output] |
...

## Detailed Results

### Scenario 1: [Name]
**Verification**: `[command]`
**Exit Code**: [0 or N]
**Output**:
\`\`\`
[actual output]
\`\`\`
**Result**: PASS / FAIL

[Continue for all scenarios]

## TDD Cycle Summary
| Phase | Status |
|-------|--------|
| RED (tests) | '  Completed |
| GREEN (implement) | '  Completed |
| REFACTOR | '  Completed |
| QA (verify) | PASS/FAIL |

## Issues Found
[If any failures, describe each issue]

## Status: PASS / FAIL
```

Step 5: Handle Outcome

If ALL pass (Score = 100%):
```bash
git add artifacts/step-2-qa-report.md
git commit -m "qa(step-2): all scenarios passed"
```

If ANY fail (Score < 100%):
Inject fix phases:

```bash
PROGRESS_FILE=$(find .claude/sprints -name "PROGRESS.yaml" -type f 2>/dev/null | head -1)

yq -i '
  (.phases[] | select(.steps) | .steps[] | select(.status == "in-progress") | .phases) +=
```

```
                  [
                    {
                      "id": "fix",
                      "status": "pending",
                      "prompt": "Fix failing scenarios for step-2.\n\nRead: artifacts/step-2-qa-report.md\n\n1. Review each FAIL scenario\n2. Fix implementation to pass\n3. Run verification to confirm\n4. Run unit tests to ensure no regression\n5. Commit fixes"
                    },
                    {
                      "id": "reverify",
                      "status": "pending",
                      "prompt": "Re-verify all scenarios after fixes.\n\nRun ALL verifications from artifacts/step-2-gherkin.md\nRun ALL unit tests.\n\nIf ALL pass: Update QA report with PASS status.\nIf ANY fail: Inject another fix phase."
                    }
                  ]
                ' "$PROGRESS_FILE"

                yq -i '
                  (.. | select(.id == "qa" and .status == "in-progress")) |=
                  (.status = "failed" | .error = "QA failed - fix phase injected")
                ' "$PROGRESS_FILE"
                ```
```

verify [completed]

Final integration verification for the step.

## Your Task
Step 3: Add visual progress chart to PDF.

Tasks:
- Generate visual chart showing sprint progress (e.g., pie chart, progress bar, or timeline)
- Chart should display: completed vs pending vs failed steps
- Consider adding timeline visualization if timestamps available
- Ensure chart renders cleanly in PDF format
- Use charting library compatible with PDF generation (e.g., chart.js with canvas, SVG-based)

Output: PDF now includes visual progress chart.

## Context
Read: context/_shared-context.md (build commands)
Read: artifacts/step-2-qa-report.md (should show PASS)

## Step 1: Full Test Suite
Run all tests (not just this step's tests):
```bash
npm test
```

Verify no regressions in other tests.

Step 2: Build Verification
```bash
npm run build
npm run typecheck
npm run lint
```

Step 3: Integration Check
- Check imports resolve correctly
- Verify no circular dependencies
- Ensure types are compatible

Step 4: Final Commit
```bash
git add -A
git status
git diff --cached --quiet || git commit -m "verify(step-2): integration verified"
```

Step Complete
The TDD cycle for this step is complete:
- ' RED: Tests written first
- ' GREEN: Minimal implementation
- ' REFACTOR: Code cleaned up
- ' QA: All scenarios pass
- ' VERIFY: Integration confirmed

step-3 [completed]

Step 4: Create command/skill for PDF export.

Tasks:
```

- Add new command or extend existing command with --pdf flag
- Command should accept sprint path as argument
- Output PDF to sprint's artifacts/ directory
- Add proper error handling and user feedback
- Ensure command works from CLI

Output: Working /export-pdf command or --pdf flag on existing command.

context [completed]

Prepare context for TDD development.

Step 0: Ensure Working in Worktree (if applicable)

```bash
# If trees directory exists, switch to worktree
if [ -d "trees/2026-01-21_issue-13-pdf-export" ]; then
  cd trees/2026-01-21_issue-13-pdf-export
fi
pwd
git branch --show-current
```

Step 1: Read Shared Context

Read: context/_shared-context.md (project patterns, test patterns, commands)

Read: context/sprint-plan.md (how this step fits in the sprint)

Step 2: Understand the Step

Step 4: Create command/skill for PDF export.

Tasks:

- Add new command or extend existing command with --pdf flag
- Command should accept sprint path as argument
- Output PDF to sprint's artifacts/ directory
- Add proper error handling and user feedback
- Ensure command works from CLI

Output: Working /export-pdf command or --pdf flag on existing command.

Output

Confirm understanding of:

- Test patterns and locations
- Build/test commands
- The step requirements

red [completed]

Write tests and gherkin scenarios BEFORE any implementation.

Your Task

Step 4: Create command/skill for PDF export.

Tasks:

- Add new command or extend existing command with --pdf flag
- Command should accept sprint path as argument
- Output PDF to sprint's artifacts/ directory
- Add proper error handling and user feedback
- Ensure command works from CLI

Output: Working /export-pdf command or --pdf flag on existing command.

Shared Context

Read: context/_shared-context.md (project patterns, test patterns, commands)

Read: context/sprint-plan.md (how this step fits in the sprint)

TDD Rule: RED Phase

In this phase, you ONLY write tests. No implementation code.

Tests should FAIL when run (because implementation doesn't exist yet).

Instructions

Part 1: Write Gherkin Scenarios (4-8 scenarios)

Create scenarios that define the expected behavior:

```gherkin
Scenario: [Descriptive name]
  Given [precondition or context]
  When [action taken]
  Then [expected outcome]
```

Verification: `[shell command that exits 0 on success, non-zero on failure]`

Pass: Exit code = 0 !' Score 1

Fail: Exit code "` 0 !' Score 0
```

### Part 2: Write Unit Tests
Create test files that exercise the expected functionality:

1. Determine test file location (from _shared-context.md patterns)
2. Write test cases that match gherkin scenarios
3. Include edge cases and error handling tests
4. Tests should be runnable but FAIL (code doesn't exist)

## Test File Template
```typescript
// Filename follows project convention from _shared-context.md
import { describe, it, expect } from '[test-framework]';

describe('[Feature under test]', () => {
  describe('[Scenario group]', () => {
    it('should [expected behavior from gherkin]', () => {
      // Arrange - Given
      // Act - When
      // Assert - Then
    });

    it('should [another expected behavior]', () => {
      // ...
    });
  });

  describe('edge cases', () => {
    it('should handle [edge case]', () => {
      // ...
    });
  });

  describe('error handling', () => {
    it('should throw when [error condition]', () => {
      // ...
    });
  });
});
```

Output
Create: artifacts/step-3-gherkin.md

```markdown
# Gherkin Scenarios: step-3

## Step Task
Step 4: Create command/skill for PDF export.

Tasks:
- Add new command or extend existing command with --pdf flag
- Command should accept sprint path as argument
- Output PDF to sprint's artifacts/ directory
- Add proper error handling and user feedback
- Ensure command works from CLI

Output: Working /export-pdf command or --pdf flag on existing command.


## Success Criteria
All scenarios must pass (score = 1) for the step to be complete.
Total scenarios: [N]
Required score: [N]/[N]

---

## Scenario 1: [Name]
[Full gherkin with verification]

---

## Scenario 2: [Name]
[Continue for 4-8 scenarios]

---

## Unit Test Coverage

| Test File | Test Cases | Scenarios Covered |
|---|---|---|
| [path] | [count] | 1, 2, 3 |

## RED Phase Verification
Tests are expected to FAIL at this point:
\`\`\`bash
npm test -- --testPathPattern="[pattern]"
# Expected: FAIL (no implementation yet)
\`\`\`
```

Commit (Tests Only)
```bash
git add [test files] artifacts/step-3-gherkin.md
git commit -m "test(step-3): add failing tests [RED]"
```

Important
- Do NOT write any implementation code in this phase
- Tests SHOULD fail - that's the point of RED
- Each gherkin scenario should have corresponding unit test(s)

green-context [completed]

Gather step-specific context for implementation.

Your Task
Step 4: Create command/skill for PDF export.

Tasks:
- Add new command or extend existing command with --pdf flag
- Command should accept sprint path as argument
- Output PDF to sprint's artifacts/ directory
- Add proper error handling and user feedback
- Ensure command works from CLI

Output: Working /export-pdf command or --pdf flag on existing command.

Shared Context
Read: context/_shared-context.md (project-wide patterns)
Read: artifacts/step-3-gherkin.md (scenarios to implement)

Instructions
Research the specific code areas this step will touch:

1. **Related Existing Code**
   - Find similar implementations to follow as patterns
   - Identify modules to import from
   - Note interfaces/types to implement or extend

2. **Dependencies**
   - Internal module imports needed
   - External packages needed
   - Version constraints

3. **Integration Points**
   - How will this code be called/used?
   - What modules will import from this?
   - Existing tests to reference

4. **Patterns for Implementation**
   - Error handling patterns
   - Naming conventions
   - File organization

Output
Create: context/step-3-context.md

```markdown
# Step Context: step-3

## Task
Step 4: Create command/skill for PDF export.

Tasks:
- Add new command or extend existing command with --pdf flag
- Command should accept sprint path as argument
- Output PDF to sprint's artifacts/ directory
- Add proper error handling and user feedback
- Ensure command works from CLI

Output: Working /export-pdf command or --pdf flag on existing command.

## Implementation Plan
Based on gherkin scenarios, implement in this order:
1. [First thing to implement]
2. [Second thing]
...

## Related Code Patterns

### Pattern from: [path]
\`\`\`typescript
// Key pattern to follow
[relevant code snippet]
\`\`\`

## Required Imports
### Internal
- `[module]`: [what to import]

### External
- `[package]`: [what to import]

## Types/Interfaces to Use
\`\`\`typescript
// From [source]
interface [Name] {
  [relevant fields]
}
\`\`\`

## Integration Points
- Called by: [consumers]
- Calls: [dependencies]

## Files to Create/Modify
| File | Action | Purpose |
|------|--------|---------|
| [path] | Create | [purpose] |
| [path] | Modify | [changes] |
```

Commit
```bash
git add context/step-3-context.md
git commit -m "context(step-3): gather implementation context"
```

green [completed]
Implement MINIMAL code to make tests pass.

Your Task
Step 4: Create command/skill for PDF export.

Tasks:
- Add new command or extend existing command with --pdf flag
- Command should accept sprint path as argument
- Output PDF to sprint's artifacts/ directory
- Add proper error handling and user feedback
- Ensure command works from CLI

Output: Working /export-pdf command or --pdf flag on existing command.

TDD Rule: GREEN Phase
Write the simplest code that makes tests pass. No more, no less.
"Make it work" - not beautiful, just working.

MUST READ BEFORE IMPLEMENTING
Read in order:
1. context/_shared-context.md (project patterns)
2. artifacts/step-3-gherkin.md (scenarios to satisfy)
3. context/step-3-context.md (specific patterns, imports)

Implementation Rules

Follow the Tests
Your implementation MUST satisfy:
- ALL gherkin scenario verifications
```

- ALL unit tests written in RED phase

Minimal Implementation
- Write the simplest code that passes tests
- Don't add features not covered by tests
- Don't optimize prematurely
- Don't add "nice to have" abstractions

Follow the Context
- Use patterns from context file
- Import from specified modules
- Follow naming conventions

Run Tests Frequently
```bash
# Run tests as you implement
npm test -- --testPathPattern="[pattern]" --watch
```

Commits
Make atomic commits for logical units:
```bash
git add [files]
git commit -m "feat(step-3): [what now works] [GREEN]"
```

Success Criteria
Before marking complete:
- [ ] All unit tests pass
- [ ] All gherkin verifications should pass
- [ ] No TypeScript errors
- [ ] Code follows context patterns

Run Final Test
```bash
npm test -- --testPathPattern="[pattern]"
# Expected: PASS (all tests green)
```

Important
- Tests MUST pass before moving to REFACTOR
- Don't refactor yet - that's the next phase
- "Quick and dirty" is OK - we'll clean up next

refactor [completed]

Refactor implementation while keeping tests green.

Your Task
Step 4: Create command/skill for PDF export.

Tasks:
- Add new command or extend existing command with --pdf flag
- Command should accept sprint path as argument
- Output PDF to sprint's artifacts/ directory
- Add proper error handling and user feedback
- Ensure command works from CLI

Output: Working /export-pdf command or --pdf flag on existing command.

TDD Rule: REFACTOR Phase
"Make it right" - improve code quality without changing behavior.
Tests MUST stay green throughout refactoring.

Context
Read: context/step-3-context.md (target patterns)
Read: artifacts/step-3-gherkin.md (scenarios that must stay passing)

Refactoring Checklist

Code Quality
- [ ] Remove any duplication
- [ ] Improve naming (variables, functions, files)
- [ ] Extract helper functions if beneficial
- [ ] Simplify complex conditionals
- [ ] Add necessary type annotations

Pattern Conformance
- [ ] Follows project conventions from context
- [ ] Error handling matches patterns
- [ ] File organization matches patterns

Clean Code
- [ ] Remove dead code
- [ ] Remove console.log/debug statements
- [ ] Fix any linting issues
- [ ] Add JSDoc for public APIs only

Refactoring Rules

Run Tests After Each Change
```bash
npm test -- --testPathPattern="[pattern]"
```
If tests fail, REVERT the change immediately.

Small Steps
- One refactoring at a time
- Commit after each successful refactoring
- Don't refactor multiple things at once

Don't Change Behavior
- No new features
- No API changes (unless fixing test file imports)
- Same inputs !' same outputs

Commits
```bash
git add [files]
git commit -m "refactor(step-3): [improvement made]"
```

Lint and Type Check
```bash
npm run lint -- [files]
npm run typecheck
```

Fix any issues found.

Final Verification
```bash
npm test -- --testPathPattern="[pattern]"
npm run lint
npm run typecheck
```

All must pass before completing this phase.

qa [completed]

Verify implementation against gherkin scenarios.

Context
Read: artifacts/step-3-gherkin.md (scenarios with verification commands)

Step 1: Run Unit Tests
```bash
npm test -- --testPathPattern="[pattern]"
```

Record results: total tests, passed, failed.

Step 2: Run Each Gherkin Verification

For EACH scenario in the gherkin file:
1. Run the verification command
2. Record result: 1 (pass) or 0 (fail)
3. If fail, capture error output

Step 3: Calculate Score

Score = (passed scenarios) / (total scenarios)
- Score = 1.0 (all pass) !' PASS
- Score < 1.0 (any fail) !' FAIL

Step 4: Generate QA Report

Create: artifacts/step-3-qa-report.md

```markdown
# QA Report: step-3
```

```
## Summary
- Gherkin Scenarios: [N] total, [N] passed, [N] failed
- Gherkin Score: [X]/[N] = [percentage]%
- Unit Tests: [N] total, [N] passed, [N] failed

## Unit Test Results
\`\`\`
[test output]
\`\`\`

## Gherkin Verification Results

| # | Scenario | Result | Details |
|---|----------|--------|---------|
| 1 | [Name] | PASS/FAIL | [output] |
| 2 | [Name] | PASS/FAIL | [output] |
...

## Detailed Results

### Scenario 1: [Name]
**Verification**: `[command]`
**Exit Code**: [0 or N]
**Output**:
\`\`\`
[actual output]
\`\`\`
**Result**: PASS / FAIL

[Continue for all scenarios]

## TDD Cycle Summary
| Phase | Status |
|-------|--------|
| RED (tests) | '  Completed |
| GREEN (implement) | '  Completed |
| REFACTOR | '  Completed |
| QA (verify) | PASS/FAIL |

## Issues Found
[If any failures, describe each issue]

## Status: PASS / FAIL
```

Step 5: Handle Outcome

If ALL pass (Score = 100%):
```bash
git add artifacts/step-3-qa-report.md
git commit -m "qa(step-3): all scenarios passed"
```

If ANY fail (Score < 100%):
Inject fix phases:

```bash
PROGRESS_FILE=$(find .claude/sprints -name "PROGRESS.yaml" -type f 2>/dev/null | head -1)

yq -i '
  (.phases[] | select(.steps) | .steps[] | select(.status == "in-progress") | .phases) +=
  [
    {
      "id": "fix",
      "status": "pending",
      "prompt": "Fix failing scenarios for step-3.\n\nRead: artifacts/step-3-qa-report.md\n\n1. Review each FAIL scenario\n2. Fix implementation to pass\n3. Run verification to confirm\n4. Run unit tests to ensure no regression\n5. Commit fixes"
    },
    {
      "id": "reverify",
      "status": "pending",
      "prompt": "Re-verify all scenarios after fixes.\n\nRun ALL verifications from artifacts/step-3-gherkin.md\nRun ALL unit tests.\n\nIf ALL pass: Update QA report with PASS status.\nIf ANY fail: Inject another fix phase."
    }
  ]
' "$PROGRESS_FILE"

yq -i '
  (.. | select(.id == "qa" and .status == "in-progress")) |=
  (.status = "failed" | .error = "QA failed - fix phase injected")
```

' "$PROGRESS_FILE"
```

verify [completed]

Final integration verification for the step.

## Your Task
Step 4: Create command/skill for PDF export.

Tasks:
- Add new command or extend existing command with --pdf flag
- Command should accept sprint path as argument
- Output PDF to sprint's artifacts/ directory
- Add proper error handling and user feedback
- Ensure command works from CLI

Output: Working /export-pdf command or --pdf flag on existing command.

## Context
Read: context/_shared-context.md (build commands)
Read: artifacts/step-3-qa-report.md (should show PASS)

## Step 1: Full Test Suite
Run all tests (not just this step's tests):
```bash
npm test
```

Verify no regressions in other tests.

Step 2: Build Verification
```bash
npm run build
npm run typecheck
npm run lint
```

Step 3: Integration Check
- Check imports resolve correctly
- Verify no circular dependencies
- Ensure types are compatible

Step 4: Final Commit
```bash
git add -A
git status
git diff --cached --quiet || git commit -m "verify(step-3): integration verified"
```

Step Complete
The TDD cycle for this step is complete:
- ' RED: Tests written first
- ' GREEN: Minimal implementation
- ' REFACTOR: Code cleaned up
- ' QA: All scenarios pass
- ' VERIFY: Integration confirmed

step-4 [in-progress]

Step 5: Documentation and final polish.

Tasks:
- Add documentation for PDF export feature
- Update README or relevant docs
- Add usage examples
- Test end-to-end workflow
- Close GitHub Issue #13 with summary of implementation

Output: Feature complete with documentation, issue closed.

context [completed]

Prepare context for TDD development.

Step 0: Ensure Working in Worktree (if applicable)
```bash
# If trees directory exists, switch to worktree
if [ -d "trees/2026-01-21_issue-13-pdf-export" ]; then
  cd trees/2026-01-21_issue-13-pdf-export
fi
pwd
git branch --show-current
```
```

```

## Step 1: Read Shared Context
Read: context/_shared-context.md (project patterns, test patterns, commands)
Read: context/sprint-plan.md (how this step fits in the sprint)

## Step 2: Understand the Step
Step 5: Documentation and final polish.

Tasks:
- Add documentation for PDF export feature
- Update README or relevant docs
- Add usage examples
- Test end-to-end workflow
- Close GitHub Issue #13 with summary of implementation

Output: Feature complete with documentation, issue closed.


## Output
Confirm understanding of:
- Test patterns and locations
- Build/test commands
- The step requirements
red [completed]
Write tests and gherkin scenarios BEFORE any implementation.

## Your Task
Step 5: Documentation and final polish.

Tasks:
- Add documentation for PDF export feature
- Update README or relevant docs
- Add usage examples
- Test end-to-end workflow
- Close GitHub Issue #13 with summary of implementation

Output: Feature complete with documentation, issue closed.


## Shared Context
Read: context/_shared-context.md (project patterns, test patterns, commands)
Read: context/sprint-plan.md (how this step fits in the sprint)

## TDD Rule: RED Phase
In this phase, you ONLY write tests. No implementation code.
Tests should FAIL when run (because implementation doesn't exist yet).

## Instructions

### Part 1: Write Gherkin Scenarios (4-8 scenarios)
Create scenarios that define the expected behavior:

```gherkin
Scenario: [Descriptive name]
  Given [precondition or context]
  When [action taken]
  Then [expected outcome]

Verification: `[shell command that exits 0 on success, non-zero on failure]`
Pass: Exit code = 0 !' Score 1
Fail: Exit code "` 0 !' Score 0
```

Part 2: Write Unit Tests
Create test files that exercise the expected functionality:

1. Determine test file location (from _shared-context.md patterns)
2. Write test cases that match gherkin scenarios
3. Include edge cases and error handling tests
4. Tests should be runnable but FAIL (code doesn't exist)

Test File Template
```typescript
// Filename follows project convention from _shared-context.md
import { describe, it, expect } from '[test-framework]';

describe('[Feature under test]', () => {
  describe('[Scenario group]', () => {
    it('should [expected behavior from gherkin]', () => {

```
      // Arrange - Given
      // Act - When
      // Assert - Then
    });

    it('should [another expected behavior]', () => {
      // ...
    });
  });

  describe('edge cases', () => {
    it('should handle [edge case]', () => {
      // ...
    });
  });

  describe('error handling', () => {
    it('should throw when [error condition]', () => {
      // ...
    });
  });
});
```

Output
Create: artifacts/step-4-gherkin.md

```markdown
# Gherkin Scenarios: step-4

## Step Task
Step 5: Documentation and final polish.

Tasks:
- Add documentation for PDF export feature
- Update README or relevant docs
- Add usage examples
- Test end-to-end workflow
- Close GitHub Issue #13 with summary of implementation

Output: Feature complete with documentation, issue closed.


## Success Criteria
All scenarios must pass (score = 1) for the step to be complete.
Total scenarios: [N]
Required score: [N]/[N]

---

## Scenario 1: [Name]
[Full gherkin with verification]

---

## Scenario 2: [Name]
[Continue for 4-8 scenarios]

---

## Unit Test Coverage
| Test File | Test Cases | Scenarios Covered |
|-----------|------------|-------------------|
| [path] | [count] | 1, 2, 3 |

## RED Phase Verification
Tests are expected to FAIL at this point:
\`\`\`bash
npm test -- --testPathPattern="[pattern]"
# Expected: FAIL (no implementation yet)
\`\`\`
```

Commit (Tests Only)
```bash
git add [test files] artifacts/step-4-gherkin.md
git commit -m "test(step-4): add failing tests [RED]"
```

Important

- Do NOT write any implementation code in this phase
- Tests SHOULD fail - that's the point of RED
- Each gherkin scenario should have corresponding unit test(s)

green-context [completed]

Gather step-specific context for implementation.

Your Task
Step 5: Documentation and final polish.

Tasks:
- Add documentation for PDF export feature
- Update README or relevant docs
- Add usage examples
- Test end-to-end workflow
- Close GitHub Issue #13 with summary of implementation

Output: Feature complete with documentation, issue closed.

Shared Context
Read: context/_shared-context.md (project-wide patterns)
Read: artifacts/step-4-gherkin.md (scenarios to implement)

Instructions
Research the specific code areas this step will touch:

1. **Related Existing Code**
   - Find similar implementations to follow as patterns
   - Identify modules to import from
   - Note interfaces/types to implement or extend

2. **Dependencies**
   - Internal module imports needed
   - External packages needed
   - Version constraints

3. **Integration Points**
   - How will this code be called/used?
   - What modules will import from this?
   - Existing tests to reference

4. **Patterns for Implementation**
   - Error handling patterns
   - Naming conventions
   - File organization

Output
Create: context/step-4-context.md

```markdown
# Step Context: step-4

## Task
Step 5: Documentation and final polish.

Tasks:
- Add documentation for PDF export feature
- Update README or relevant docs
- Add usage examples
- Test end-to-end workflow
- Close GitHub Issue #13 with summary of implementation

Output: Feature complete with documentation, issue closed.

## Implementation Plan
Based on gherkin scenarios, implement in this order:
1. [First thing to implement]
2. [Second thing]
...

## Related Code Patterns

### Pattern from: [path]
\`\`\`typescript
// Key pattern to follow
[relevant code snippet]
\`\`\`

## Required Imports
```

Internal
- `[module]`: [what to import]

External
- `[package]`: [what to import]

Types/Interfaces to Use
\`\`\`typescript
// From [source]
interface [Name] {
  [relevant fields]
}
\`\`\`

Integration Points
- Called by: [consumers]
- Calls: [dependencies]

Files to Create/Modify
File	Action	Purpose
[path]	Create	[purpose]
[path]	Modify	[changes]
```

## Commit
```bash
git add context/step-4-context.md
git commit -m "context(step-4): gather implementation context"
```

green [completed]
Implement MINIMAL code to make tests pass.

Your Task
Step 5: Documentation and final polish.

Tasks:
- Add documentation for PDF export feature
- Update README or relevant docs
- Add usage examples
- Test end-to-end workflow
- Close GitHub Issue #13 with summary of implementation

Output: Feature complete with documentation, issue closed.

TDD Rule: GREEN Phase
Write the simplest code that makes tests pass. No more, no less.
"Make it work" - not beautiful, just working.

MUST READ BEFORE IMPLEMENTING
Read in order:
1. context/_shared-context.md (project patterns)
2. artifacts/step-4-gherkin.md (scenarios to satisfy)
3. context/step-4-context.md (specific patterns, imports)

Implementation Rules

Follow the Tests
Your implementation MUST satisfy:
- ALL gherkin scenario verifications
- ALL unit tests written in RED phase

Minimal Implementation
- Write the simplest code that passes tests
- Don't add features not covered by tests
- Don't optimize prematurely
- Don't add "nice to have" abstractions

Follow the Context
- Use patterns from context file
- Import from specified modules
- Follow naming conventions

Run Tests Frequently
```bash
# Run tests as you implement
npm test -- --testPathPattern="[pattern]" --watch
```
```

Commits
Make atomic commits for logical units:
```bash
git add [files]
git commit -m "feat(step-4): [what now works] [GREEN]"
```

Success Criteria
Before marking complete:
- [ ] All unit tests pass
- [ ] All gherkin verifications should pass
- [ ] No TypeScript errors
- [ ] Code follows context patterns

Run Final Test
```bash
npm test -- --testPathPattern="[pattern]"
# Expected: PASS (all tests green)
```

Important
- Tests MUST pass before moving to REFACTOR
- Don't refactor yet - that's the next phase
- "Quick and dirty" is OK - we'll clean up next

refactor [completed]

Refactor implementation while keeping tests green.

Your Task
Step 5: Documentation and final polish.

Tasks:
- Add documentation for PDF export feature
- Update README or relevant docs
- Add usage examples
- Test end-to-end workflow
- Close GitHub Issue #13 with summary of implementation

Output: Feature complete with documentation, issue closed.

TDD Rule: REFACTOR Phase
"Make it right" - improve code quality without changing behavior.
Tests MUST stay green throughout refactoring.

Context
Read: context/step-4-context.md (target patterns)
Read: artifacts/step-4-gherkin.md (scenarios that must stay passing)

Refactoring Checklist

Code Quality
- [ ] Remove any duplication
- [ ] Improve naming (variables, functions, files)
- [ ] Extract helper functions if beneficial
- [ ] Simplify complex conditionals
- [ ] Add necessary type annotations

Pattern Conformance
- [ ] Follows project conventions from context
- [ ] Error handling matches patterns
- [ ] File organization matches patterns

Clean Code
- [ ] Remove dead code
- [ ] Remove console.log/debug statements
- [ ] Fix any linting issues
- [ ] Add JSDoc for public APIs only

Refactoring Rules

Run Tests After Each Change
```bash
npm test -- --testPathPattern="[pattern]"
```
If tests fail, REVERT the change immediately.

Small Steps
- One refactoring at a time
- Commit after each successful refactoring
- Don't refactor multiple things at once

Don't Change Behavior
- No new features
- No API changes (unless fixing test file imports)
- Same inputs !' same outputs

Commits
```bash
git add [files]
git commit -m "refactor(step-4): [improvement made]"
```

Lint and Type Check
```bash
npm run lint -- [files]
npm run typecheck
```

Fix any issues found.

Final Verification
```bash
npm test -- --testPathPattern="[pattern]"
npm run lint
npm run typecheck
```

All must pass before completing this phase.

qa [completed]

Verify implementation against gherkin scenarios.

Context
Read: artifacts/step-4-gherkin.md (scenarios with verification commands)

Step 1: Run Unit Tests
```bash
npm test -- --testPathPattern="[pattern]"
```

Record results: total tests, passed, failed.

Step 2: Run Each Gherkin Verification

For EACH scenario in the gherkin file:
1. Run the verification command
2. Record result: 1 (pass) or 0 (fail)
3. If fail, capture error output

Step 3: Calculate Score

Score = (passed scenarios) / (total scenarios)
- Score = 1.0 (all pass) !' PASS
- Score < 1.0 (any fail) !' FAIL

Step 4: Generate QA Report

Create: artifacts/step-4-qa-report.md

```markdown
# QA Report: step-4

## Summary
- Gherkin Scenarios: [N] total, [N] passed, [N] failed
- Gherkin Score: [X]/[N] = [percentage]%
- Unit Tests: [N] total, [N] passed, [N] failed

## Unit Test Results
\`\`\`
[test output]
\`\`\`

## Gherkin Verification Results

| # | Scenario | Result | Details |
|---|----------|--------|---------|
| 1 | [Name] | PASS/FAIL | [output] |
| 2 | [Name] | PASS/FAIL | [output] |
...

## Detailed Results
```

```
### Scenario 1: [Name]
**Verification**: `[command]`
**Exit Code**: [0 or N]
**Output**:
\`\`\`
[actual output]
\`\`\`
**Result**: PASS / FAIL

[Continue for all scenarios]

## TDD Cycle Summary
| Phase | Status |
|-------|--------|
| RED (tests) | '  Completed |
| GREEN (implement) | '  Completed |
| REFACTOR | '  Completed |
| QA (verify) | PASS/FAIL |

## Issues Found
[If any failures, describe each issue]

## Status: PASS / FAIL
```

Step 5: Handle Outcome

If ALL pass (Score = 100%):
```bash
git add artifacts/step-4-qa-report.md
git commit -m "qa(step-4): all scenarios passed"
```

If ANY fail (Score < 100%):
Inject fix phases:

```bash
PROGRESS_FILE=$(find .claude/sprints -name "PROGRESS.yaml" -type f 2>/dev/null | head -1)

yq -i '
  (.phases[] | select(.steps) | .steps[] | select(.status == "in-progress") | .phases) +=
  [
    {
      "id": "fix",
      "status": "pending",
      "prompt": "Fix failing scenarios for step-4.\n\nRead: artifacts/step-4-qa-report.md\n\n1. Review each FAIL scenario\n2. Fix implementation to pass\n3. Run verification to confirm\n4. Run unit tests to ensure no regression\n5. Commit fixes"
    },
    {
      "id": "reverify",
      "status": "pending",
      "prompt": "Re-verify all scenarios after fixes.\n\nRun ALL verifications from artifacts/step-4-gherkin.md\nRun ALL unit tests.\n\nIf ALL pass: Update QA report with PASS status.\nIf ANY fail: Inject another fix phase."
    }
  ]
' "$PROGRESS_FILE"

yq -i '
  (.. | select(.id == "qa" and .status == "in-progress")) |=
  (.status = "failed" | .error = "QA failed - fix phase injected")
' "$PROGRESS_FILE"
```
```

verify [in-progress]
Final integration verification for the step.

## Your Task
Step 5: Documentation and final polish.

Tasks:
- Add documentation for PDF export feature
- Update README or relevant docs
- Add usage examples
- Test end-to-end workflow
- Close GitHub Issue #13 with summary of implementation

Output: Feature complete with documentation, issue closed.


## Context
```

Read: context/_shared-context.md (build commands)
Read: artifacts/step-4-qa-report.md (should show PASS)

Step 1: Full Test Suite
Run all tests (not just this step's tests):
```bash
npm test
```

Verify no regressions in other tests.

Step 2: Build Verification
```bash
npm run build
npm run typecheck
npm run lint
```

Step 3: Integration Check
- Check imports resolve correctly
- Verify no circular dependencies
- Ensure types are compatible

Step 4: Final Commit
```bash
git add -A
git status
git diff --cached --quiet || git commit -m "verify(step-4): integration verified"
```

Step Complete
The TDD cycle for this step is complete:
- ' RED: Tests written first
- ' GREEN: Minimal implementation
- ' REFACTOR: Code cleaned up
- ' QA: All scenarios pass
- ' VERIFY: Integration confirmed

doc-analyze [pending]

Analyze code changes and plan documentation updates.

Step 0: Ensure Working in Worktree
```bash
cd trees/2026-01-21_issue-13-pdf-export 2>/dev/null || true
pwd
git branch --show-current  # Verify: sprint/2026-01-21_issue-13-pdf-export
```

Context
Read: context/_shared-context.md (documentation structure)
Read: context/sprint-plan.md (documentation update plan)

Step 1: Review All Code Changes
```bash
# See all changes in this sprint
git diff main..HEAD --name-only

# Get detailed diff
git diff main..HEAD -- "*.ts" "*.tsx" "*.js"
```

Step 2: Identify Documentation Impact

For each change, determine:
- New features that need documenting
- Changed behavior that needs updating
- Removed features that need removing from docs
- New/changed CLI commands
- New/changed configuration options
- New/changed API endpoints or functions

Step 3: Audit Existing Documentation
```bash
# Find all documentation files
find . -name "*.md" -path "*/docs/*" -o -name "README.md" | head -30
```

Review each relevant doc file and note:
- What's accurate and complete
- What's outdated
- What's missing

Step 4: Generate Documentation Plan
Create: artifacts/docs-update-plan.md

```markdown
# Documentation Update Plan: 2026-01-21_issue-13-pdf-export

## Code Changes Summary
| File | Change Type | Description |
|------|-------------|-------------|
| [path] | Added/Modified | [what changed] |

## Documentation Impact

### User Guide Updates
| Section | Action | Reason |
|---------|--------|--------|
| [section] | Update/Add/Remove | [why] |

### Getting Started Updates
| Section | Action | Reason |
|---------|--------|--------|
| [section] | Update/Add | [why] |

### Reference Updates
| Item | Action | Details |
|------|--------|---------|
| [command/function] | Update/Add | [changes] |

## New Documentation Needed
- [ ] [New doc 1]: [purpose]
- [ ] [New doc 2]: [purpose]

## Files to Update
| File | Updates Needed |
|------|----------------|
| [path] | [specific updates] |

## Verification Plan
- [ ] All code examples tested
- [ ] All commands verified
- [ ] All links checked
```

Commit
```bash
git add artifacts/docs-update-plan.md
git commit -m "docs(plan): documentation update analysis"
```

doc-user-guide [pending]
Update user-facing documentation based on analysis.

Context

Read: artifacts/docs-update-plan.md (what needs updating)
Read: context/_shared-context.md (documentation structure)

User Guide Principles
1. **Task-oriented**: Organize by what users want to DO
2. **Progressive**: Simple !' Advanced
3. **Example-rich**: Every feature needs an example
4. **Scannable**: Use headers, lists, tables

Step 1: Locate or Create User Guide
The user guide should be at: `docs/USER-GUIDE.md` or `docs/user-guide/`

If it doesn't exist, create it using the template below.

Step 2: Update Each Section

For each item in docs-update-plan.md's "User Guide Updates":

Adding New Features
```markdown
## [Feature Name]

[One paragraph explaining WHAT it does and WHY you'd use it]

### Quick Example
\`\`\`bash
# Show the simplest possible example
[command or code]
\`\`\`

### How It Works
[Explain the feature in more detail]

### Examples

#### Basic Usage
\`\`\`bash
[example]
\`\`\`

#### Advanced Usage
\`\`\`bash
[example with options]
\`\`\`

### Tips
- [Helpful tip 1]
- [Helpful tip 2]
```

Updating Existing Features
- Locate the existing section
- Update examples if behavior changed
- Update descriptions if functionality changed
- Add new subsections for new capabilities

Removing Features
- Remove the section entirely
- Or add deprecation notice if replacing

Step 3: Verify Examples Work
Run each code example in the documentation to ensure it works:
```bash
# Test each example manually
```

[example command from docs]
```

## Step 4: Commit Updates
```bash
git add docs/
git commit -m "docs(user-guide): update for 2026-01-21_issue-13-pdf-export changes"
```

doc-getting-started [pending]
Update getting started and quickstart documentation.

Context
Read: artifacts/docs-update-plan.md (what needs updating)
Read: context/_shared-context.md (documentation structure)

Getting Started Principles
1. **Fast**: Get to "Hello World" in < 5 steps
2. **Complete**: All prerequisites clearly listed
3. **Copy-paste**: Every command should work as written
4. **No assumptions**: Don't assume prior knowledge

Step 1: Locate or Create Getting Started
Should be at: `docs/getting-started/` or `docs/QUICKSTART.md` or in README.md

Step 2: Update Prerequisites
List everything needed before starting:
```markdown
## Prerequisites

- [Requirement 1] - [how to check/install]
- [Requirement 2] - [how to check/install]

### Version Requirements
| Tool | Minimum Version | Check Command |
|------|-----------------|---------------|
| [tool] | [version] | `[command]` |
```

Step 3: Update Installation
```markdown
## Installation

### Option 1: [Primary method]
\`\`\`bash
[installation commands]
\`\`\`

### Option 2: [Alternative method]
\`\`\`bash
[alternative commands]
\`\`\`

### Verify Installation
\`\`\`bash
[verification command]
# Expected output: [what they should see]
\`\`\`
```

Step 4: Update First Steps
```markdown
## Your First [Thing]

### Step 1: [Action]
\`\`\`bash
[command]
\`\`\`

You should see:
\`\`\`
[expected output]
\`\`\`

### Step 2: [Action]
[Continue with clear steps]

### Step 3: [Action]
[Each step should be verifiable]

## Success!
You've now [accomplished goal]. Next steps:
- [Link to User Guide]
- [Link to next tutorial]
```

Step 5: Update README Quick Example
If README.md has a quick example section, ensure it's current:
- Examples should work with latest code
- Version numbers should be accurate
- Links should be valid

Step 6: Test the Flow
Follow the getting started guide yourself:
1. Start from a fresh state
2. Run each command exactly as written
3. Verify expected outputs match
4. Fix any issues found

Step 7: Commit Updates
```bash
git add docs/ README.md
git commit -m "docs(getting-started): update onboarding for 2026-01-21_issue-13-pdf-export"
```

doc-reference [pending]
Update technical reference and API documentation.

Context
Read: artifacts/docs-update-plan.md (what needs updating)
Read: context/_shared-context.md (documentation structure)

Reference Documentation Principles
1. **Complete**: Document ALL public APIs
2. **Accurate**: Generated from code when possible
3. **Structured**: Consistent format for all items
4. **Searchable**: Easy to find specific items

Step 1: Locate or Create Reference Docs
Should be at: `docs/reference/` or `docs/api/`

Step 2: Update Command Reference
For CLI commands, use this format:

```markdown
## Commands Reference

### [command-name]

[One-line description]

#### Synopsis
\`\`\`
[command] [options] [arguments]
\`\`\`

#### Description
[Detailed description of what the command does]

#### Options
| Option | Type | Default | Description |
|--------|------|---------|-------------|
| `--option` | string | none | [description] |
| `-f, --flag` | boolean | false | [description] |

#### Arguments
| Argument | Required | Description |
|----------|----------|-------------|
| `<name>` | Yes | [description] |
| `[path]` | No | [description] |

#### Examples
\`\`\`bash
# Basic usage
[command] [basic-args]

# With options
[command] --option value

# Advanced usage
[command] --flag -o value
\`\`\`

#### See Also
- [Related command 1]
- [Related command 2]
```

Step 3: Update API Reference
For functions/classes, use this format:

```markdown
## API Reference

### [FunctionName]

\`\`\`typescript
function name(param: Type): ReturnType
\`\`\`

[Description of what the function does]

#### Parameters
| Parameter | Type | Required | Description |
|-----------|------|----------|-------------|
| `param` | `Type` | Yes | [description] |

#### Returns
`ReturnType` - [description of return value]

#### Throws
- `ErrorType` - [when this error is thrown]

#### Example
\`\`\`typescript
const result = name(value);
console.log(result); // [expected output]
\`\`\`
```

Step 4: Update Configuration Reference
For config options:

```markdown
## Configuration Reference

### [config-file-name]

#### Schema
\`\`\`yaml
# Full schema with comments
option: value  # description
nested:
  key: value   # description
\`\`\`

#### Options

##### `option`
- **Type**: string
- **Required**: Yes/No
- **Default**: `value`
- **Description**: [what this option does]
- **Example**: `option: "example"`
```

Step 5: Update Type Definitions
If there are TypeScript types that users need to know:

```markdown
## Type Definitions

### [TypeName]
\`\`\`typescript
interface TypeName {
  property: string;
  optional?: number;
}
\`\`\`

| Property | Type | Required | Description |
|----------|------|----------|-------------|
| `property` | `string` | Yes | [description] |
```

Step 6: Verify Reference Accuracy
- Check that all documented APIs actually exist
- Verify parameter names and types match code
- Test example code snippets

Step 7: Commit Updates
```bash
git add docs/reference/
git commit -m "docs(reference): update API and command reference for 2026-01-21_issue-13-pdf-export"
```
```

doc-validate [pending]

Validate all documentation updates.

Context
Read: artifacts/docs-update-plan.md (original plan)

Step 1: Completeness Check
Verify all planned updates were made:
```bash
# Show all doc changes in this sprint
git diff main..HEAD -- "*.md" --stat
```

Cross-reference with docs-update-plan.md:
- [ ] All User Guide updates complete
- [ ] All Getting Started updates complete
- [ ] All Reference updates complete
- [ ] All new docs created

Step 2: Link Validation
Check all internal links work:
```bash
# Find all markdown links
grep -rh "\[.*\](.*\.md)" docs/ | head -30
```

Verify each linked file exists.

Step 3: Code Example Validation
Test each code example in the documentation:
- Run CLI examples
- Compile TypeScript examples
- Execute JavaScript examples

Document any failures.

Step 4: Consistency Check
- Consistent formatting across all docs
- Consistent terminology
- Consistent example style
- Version numbers match

Step 5: Generate Validation Report
Create: artifacts/docs-validation-report.md

```markdown
# Documentation Validation Report: 2026-01-21_issue-13-pdf-export

## Completeness
| Planned Update | Status | Notes |
|---|---|---|
| [update 1] | DONE/MISSING | [notes] |

## Link Validation
| Link | Target | Status |
|---|---|---|
| [link text] | [file] | OK/BROKEN |

## Code Example Validation
| File | Example | Status | Output |
|---|---|---|---|
| [path] | Line [N] | PASS/FAIL | [output] |
```

Consistency Check
- [ ] Formatting consistent
- [ ] Terminology consistent
- [ ] Examples consistent
- [ ] Versions accurate

Issues Found
[List any issues, or "None"]

Documentation Files Updated
File	Lines Changed	Summary
[path]	+[N]/-[N]	[summary]

Overall Status: PASS / FAIL
```

## Step 6: Fix Any Issues
If validation found issues:
- Fix broken links
- Update failed examples
- Correct inconsistencies

## Step 7: Final Commit
```bash
git add artifacts/docs-validation-report.md
git add docs/  # Any fixes
git commit -m "docs(validate): documentation verified for 2026-01-21_issue-13-pdf-export"
```

Important
Documentation must pass validation before sprint can complete.
All code examples must work with the actual code.

final-qa [pending]

Comprehensive Quality Assurance for the entire sprint.

Step 0: Ensure Working in Worktree
```bash
cd trees/2026-01-21_issue-13-pdf-export 2>/dev/null || true
pwd
git branch --show-current  # Verify: sprint/2026-01-21_issue-13-pdf-export
```

Context
Read: context/_shared-context.md for build/test commands
Read: context/sprint-plan.md for success criteria

Step 1: Full Build Verification
Run ALL build checks:
```bash
npm run build
npm run typecheck
npm run lint
```

Record each result in the QA report.

Step 2: Complete Test Suite
Run the full test suite:
```bash
npm test
```
```

Record test results, coverage, and any failures.

Step 3: Review All Step QA Reports
Read all artifacts/step-*-qa-report.md files:
- Verify all steps show PASS status
- Calculate total gherkin score
- Consolidate warnings or notes

Step 4: Documentation Verification
Verify documentation was updated:
```bash
# Check for documentation changes
git diff main..HEAD -- "*.md" --stat
```

Verify:
- User guide reflects new features
- Getting started is accurate
- Reference material is complete
- Examples work correctly

Step 5: Integration Verification
- Check modules properly import each other
- Verify no circular dependencies
- Test end-to-end flow if applicable

Step 6: Regression Check
```bash
git diff main...HEAD --stat
```

Verify:
- No unintended changes
- All modified files are expected
- No debug code left in

Step 7: Generate Sprint QA Report
Create: artifacts/sprint-qa-report.md

```markdown
# Sprint QA Report: 2026-01-21_issue-13-pdf-export

## Build Verification
| Check | Result | Output |
|-------|--------|--------|
| Build | PASS/FAIL | [summary] |
| TypeCheck | PASS/FAIL | [summary] |
| Lint | PASS/FAIL | [summary] |

## Test Suite
| Metric | Value |
|--------|-------|
| Tests Run | [count] |
| Passed | [count] |
| Failed | [count] |
| Coverage | [percentage] |

## Gherkin Scenario Summary
| Step | Total | Passed | Score |
|------|-------|--------|-------|
| step-0 | [N] | [N] | 100% |
| step-1 | [N] | [N] | 100% |
| **Total** | [N] | [N] | **100%** |
```

Documentation Status
Document	Status	Changes
User Guide	PASS/FAIL	[summary]
Getting Started	PASS/FAIL	[summary]
Reference	PASS/FAIL	[summary]

Integration Verification
- [ ] Modules import correctly
- [ ] No circular dependencies
- [ ] End-to-end flow works

Overall Status: PASS / FAIL
```

## Step 8: Handle Outcome

### If PASS:
```bash
git add artifacts/sprint-qa-report.md
git commit -m "qa: sprint-level verification passed"
```

If FAIL:
- Document specific failures
- Set status to needs-human with details

summary [pending]

Generate sprint summary with TDD and documentation highlights.

Context
Read: context/sprint-plan.md for original goals
Read: artifacts/sprint-qa-report.md for verification results
Read: All artifacts/step-*-qa-report.md for step details

Step 1: Collect Commit History
```bash
git log main..HEAD --oneline --no-decorate
```

Step 2: Collect File Changes
```bash
git diff main..HEAD --stat
```

Step 3: Generate Sprint Summary
Create: artifacts/sprint-summary.md

```markdown
# Sprint Summary: 2026-01-21_issue-13-pdf-export

## What Was Accomplished

### Step 0: [Step title]
**TDD Cycle**:
- Tests written: [count]
- Gherkin scenarios: [count], all passing

**Implementation**:
- [Key accomplishment 1]
- [Key accomplishment 2]

**Files**: [list]

[Continue for each step]

## Test Coverage Summary
| Metric | Before | After | Delta |
|--------|--------|-------|-------|
| Tests | [N] | [N] | +[N] |
| Gherkin | [N] | [N] | +[N] |
| Coverage | [%] | [%] | +[%] |

## Documentation Updates
| Document | Change |
|----------|--------|
| [path] | [what changed] |

## Files Changed
| File | Change Type | Description |
|------|-------------|-------------|
| [path] | Created/Modified | [description] |

## Commits Made
| Hash | Type | Message |
|------|------|---------|
| [hash] | test | [message] |
| [hash] | feat | [message] |
| [hash] | docs | [message] |

## Verification Status
- Build: PASS
- TypeCheck: PASS
- Lint: PASS
- Tests: [X/Y passed]
- Gherkin: [X/Y scenarios, 100%]
- Documentation: Updated

## Sprint Statistics
- Steps completed: [X/Y]
- Total commits: [count]
- Tests added: [count]
- Gherkin scenarios: [count]
- Files changed: [count]
```

Step 4: Commit Summary
```bash
git add artifacts/sprint-summary.md
git commit -m "docs: add sprint summary"
```

version-bump [pending]
Increment the plugin version number.

Step 0: Ensure Working in Worktree
```bash
cd trees/2026-01-21_issue-13-pdf-export 2>/dev/null || true
pwd
```

Step 1: Identify the Plugin
Determine which plugin this sprint modifies by checking:
1. The sprint name/id for plugin hints
2. Files changed in this sprint
3. The SPRINT.yaml description

```bash

```
List changed files to identify plugin
git diff main..HEAD --name-only | grep "^plugins/" | cut -d'/' -f2 | sort -u
```

Step 2: Read Current Versions
For the identified plugin (e.g., `m42-sprint`):

```bash
# Plugin's own version
cat plugins/<plugin-name>/.claude-plugin/plugin.json | grep version

# Marketplace entry version
cat .claude-plugin/marketplace.json | grep -A5 '"<plugin-name>"'
```

Step 3: Determine Version Bump Type
Based on changes in this sprint:
- **MAJOR** (X.0.0): Breaking changes, incompatible API changes
- **MINOR** (0.X.0): New features, backward-compatible additions
- **PATCH** (0.0.X): Bug fixes, small improvements

Default to MINOR for feature sprints, PATCH for bugfix sprints.

Step 4: Update Plugin Version
Update `plugins/<plugin-name>/.claude-plugin/plugin.json`:

```bash
# Example: bump from 2.0.0 to 2.1.0
# Use jq or manual edit to update version field
```

Step 5: Update Marketplace Version
Update `.claude-plugin/marketplace.json` to match:

Find the plugin entry and update its version to match plugin.json.

IMPORTANT: Both versions MUST be identical.

Step 6: Commit Version Bump
```bash
git add plugins/<plugin-name>/.claude-plugin/plugin.json
git add .claude-plugin/marketplace.json
git commit -m "chore: bump <plugin-name> version to X.Y.Z"
```

Output
- Plugin version updated in `plugins/<plugin-name>/.claude-plugin/plugin.json`
- Marketplace version updated in `.claude-plugin/marketplace.json`
- Both versions are in sync

pr-create [pending]
Push the sprint branch and create a pull request.

Context
Read: artifacts/sprint-summary.md for PR body content
Read: artifacts/sprint-qa-report.md for verification checklist
Read: context/worktree-info.md for worktree paths

Step 0: Verify Working in Worktree
```bash
# Ensure we're in the sprint worktree
pwd
git branch --show-current  # Should be: sprint/2026-01-21_issue-13-pdf-export
```

Step 1: Ensure All Changes Committed
```bash
git status
```

Step 2: Push Branch
```bash
git push -u origin sprint/2026-01-21_issue-13-pdf-export
```

Step 3: Create Pull Request
```bash
gh pr create \
  --title "Sprint: 2026-01-21_issue-13-pdf-export" \
  --body "$(cat <<'EOF'
## Summary
[Extract key points from sprint-summary.md]

## TDD Approach
- Tests written first for each step
- All gherkin scenarios pass
- [X] new tests added

## Changes
[List major changes]

## Documentation
- [ ] User Guide updated
- [ ] Getting Started updated
- [ ] Reference updated

## Verification Checklist
- [x] Build passes
- [x] TypeCheck passes
- [x] Lint passes
- [x] All tests pass
- [x] Gherkin scenarios: 100%
- [x] Documentation updated
- [x] No regressions

---
Full details in `artifacts/sprint-summary.md`

Ø>Ýê TDD Sprint with Documentation
EOF
)"
```

Step 4: Output PR URL
```bash
gh pr view --json url -q '.url'
```

Step 5: Worktree Cleanup Instructions
After the PR is merged, clean up the worktree:

```bash
# From the main repository (not the worktree)
cd ..  # Go to main repo root

# Remove the worktree
git worktree remove trees/2026-01-21_issue-13-pdf-export
```

```
# Delete the remote branch (optional, after PR merge)
git push origin --delete sprint/2026-01-21_issue-13-pdf-export

# List remaining worktrees
git worktree list
```

Note: Do NOT remove the worktree until the PR is merged!