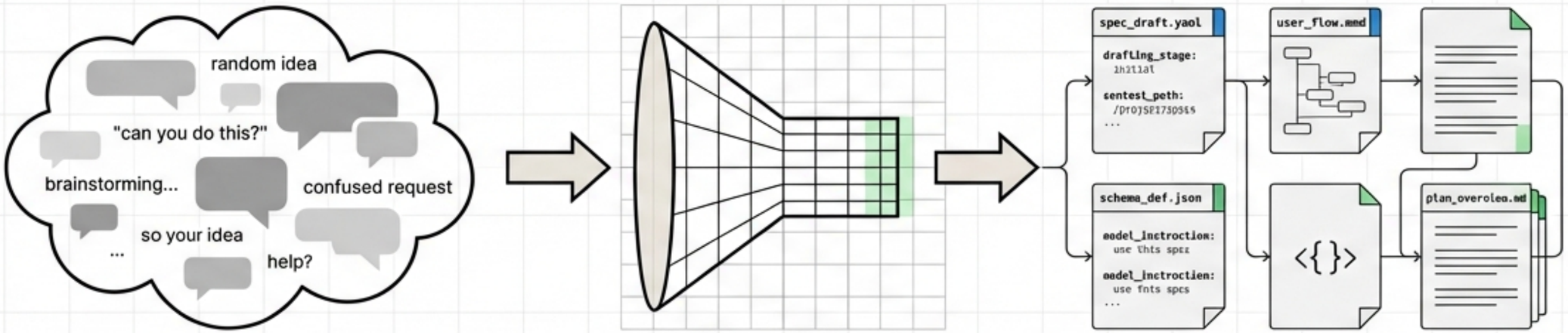
# Draft: Context-Driven Development

## A Methodology for Structured AI Engineering




**Unstructured Chat**
Raw User Intent, No Schema

**Filter**
Validation, Schema, Constraints

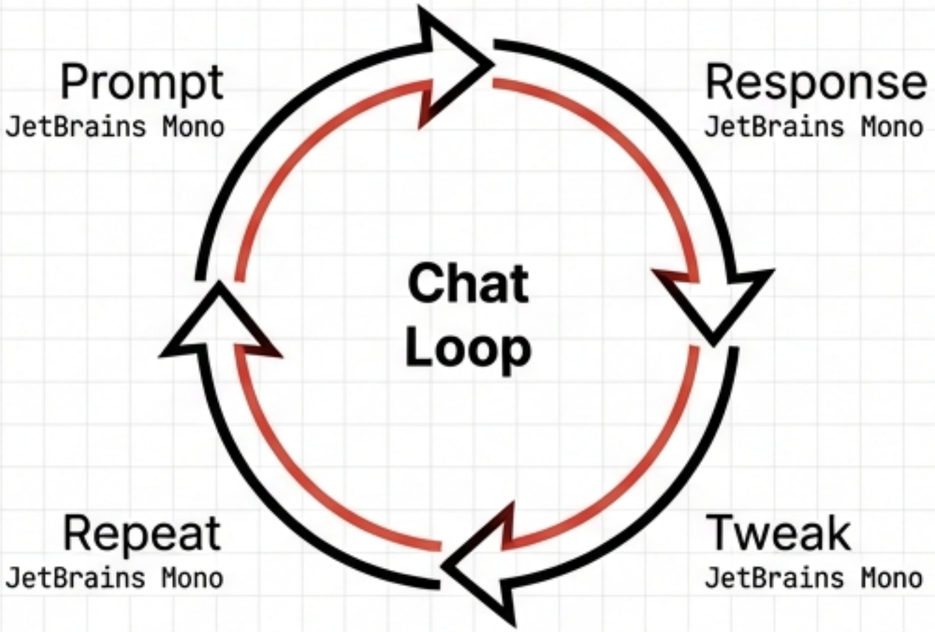
**Structured Files**
Executable Specs, Validated Plans

*"Measure twice, code once. A framework for drafting specs and plans before implementation, ensuring AI adheres to project truth rather than hallucinated assumptions."*

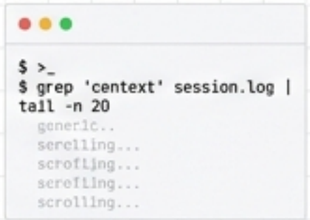
Compatible with Claude Code & Cursor

# Moving Beyond “Chatty” Development

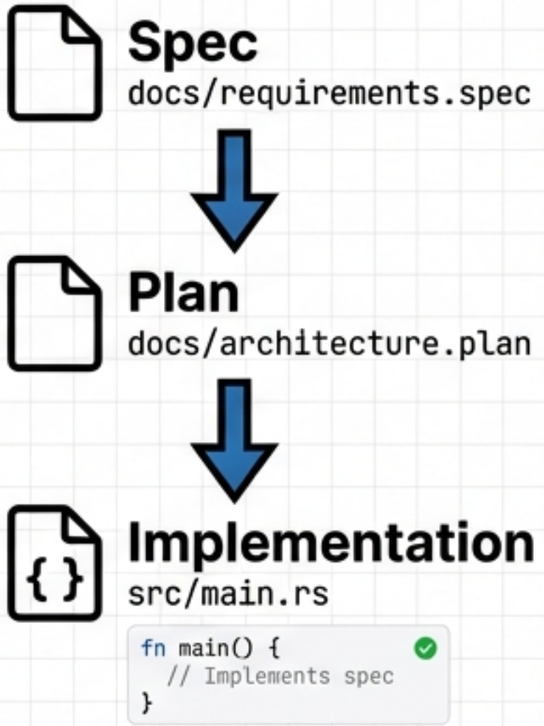
## The Problem: Ephemeral Context



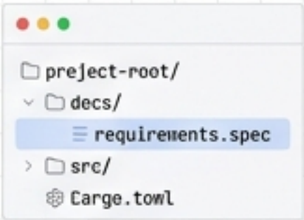

- Context lives in the scrollback buffer
- Memory loss on long sessions
- Result: Hallucinations & Architectural Drift




## The Solution: Context-Driven Development

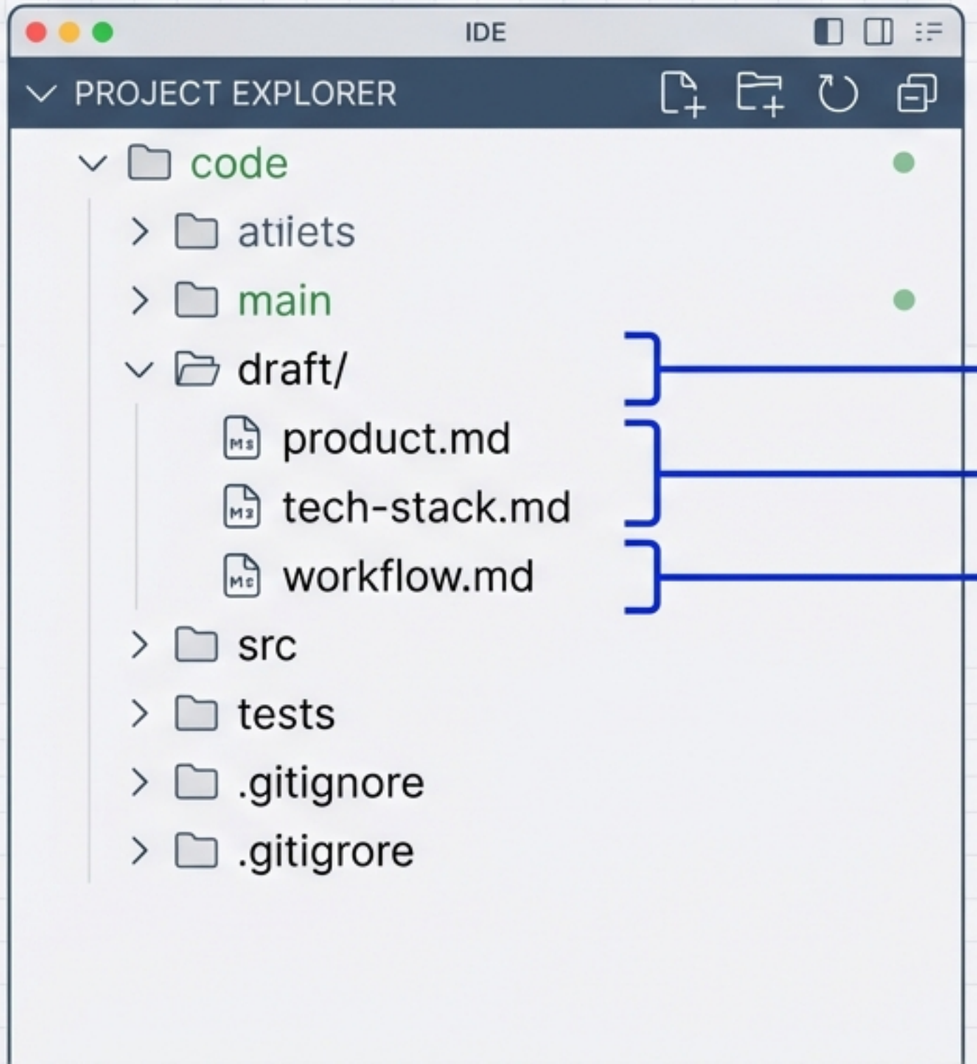



- Context anchors to persistent files
- Project Truth exists in the filesystem
- Result: Defined constraints before coding



# Establishing the Project Truth

The ‘**draft/**’ directory grounds the AI in non-negotiable reality.




**Vision & Goals.** Defines what we are building and why.

**Constraints.** Libraries, versions, and architectural choices.

**Standards.** Commit preferences, naming conventions, and TDD rules.

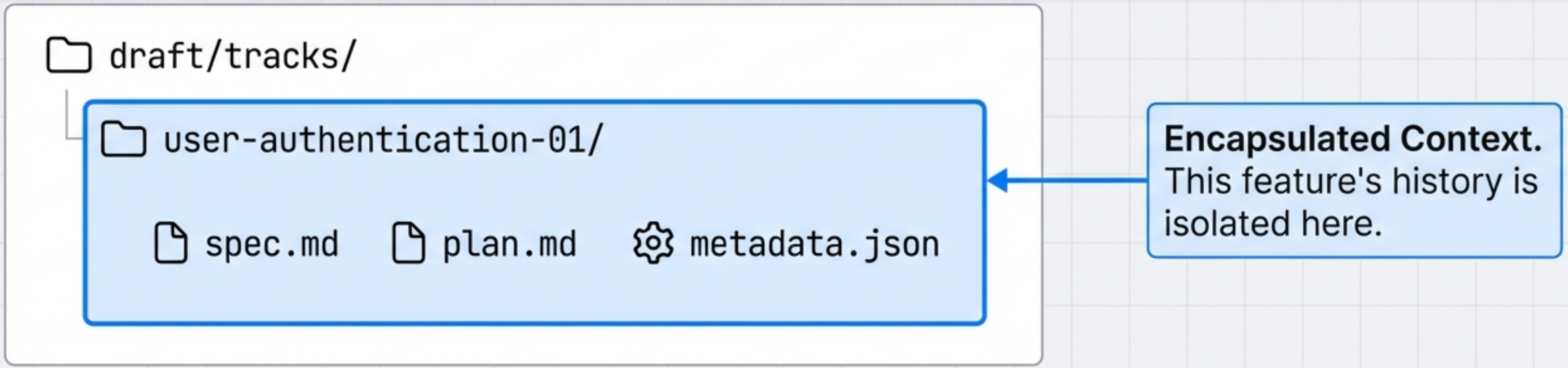
**Note**

**Global Context:** These files auto-load with every session, ensuring the AI never forgets the rules.

# The Unit of Work is the ‘Track’

Work is organized into encapsulated directories, ensuring isolation of context and clear history.

```
> /draft:new-track 'User Authentication'
```



# Phase 1: The Specification

Defining the 'What' before the 'How'

spec.md

```
# Feature: User Authentication

## Requirements
- Users must sign in via Email/Password
- Passwords must be hashed with Argon2

## Acceptance Criteria
- [ ] Successful login returns JWT
- [ ] Failed login returns 401 error
```

## The Contract

Generated automatically via /draft:new-track.

Acts as an agreement between human intent and AI execution. The AI must agree to these requirements before writing code.

# Phase 2: The Plan

## Breaking Complexity into Phases

plan.md

```
## Execution Plan

### Phase 1: Scaffolding
[x] Create user model
[ ] Set up database migration

### Phase 2: Core Logic
[ ] Implement password hashing service
[ ] Create login route handler
```

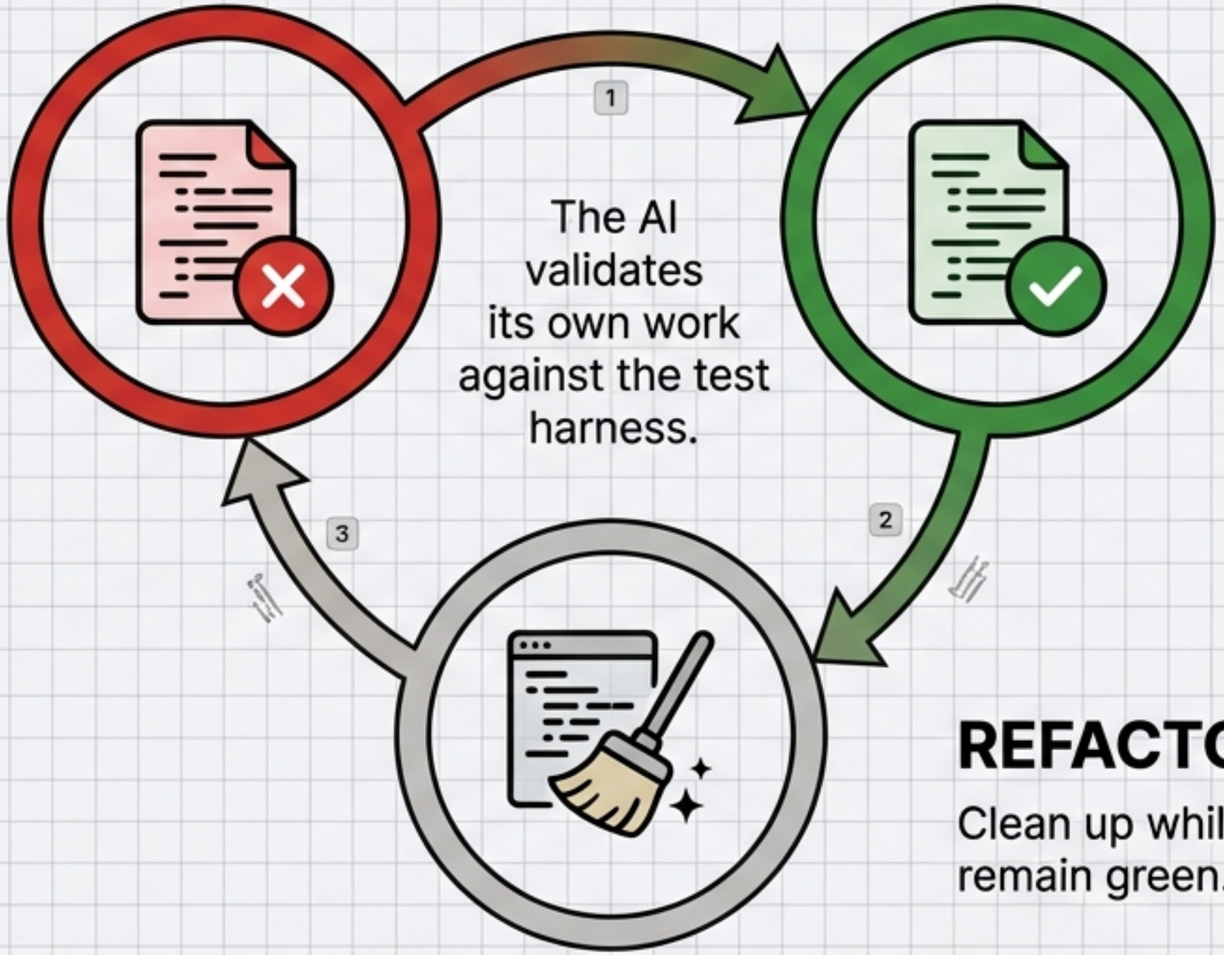
Prevents "**Code it all at once**" errors. The developer reviews this strategy before any implementation begins.

# Phase 3: Execution via TDD

`> /draft:implement`

## RED

Write a failing test based on the plan.




## GREEN

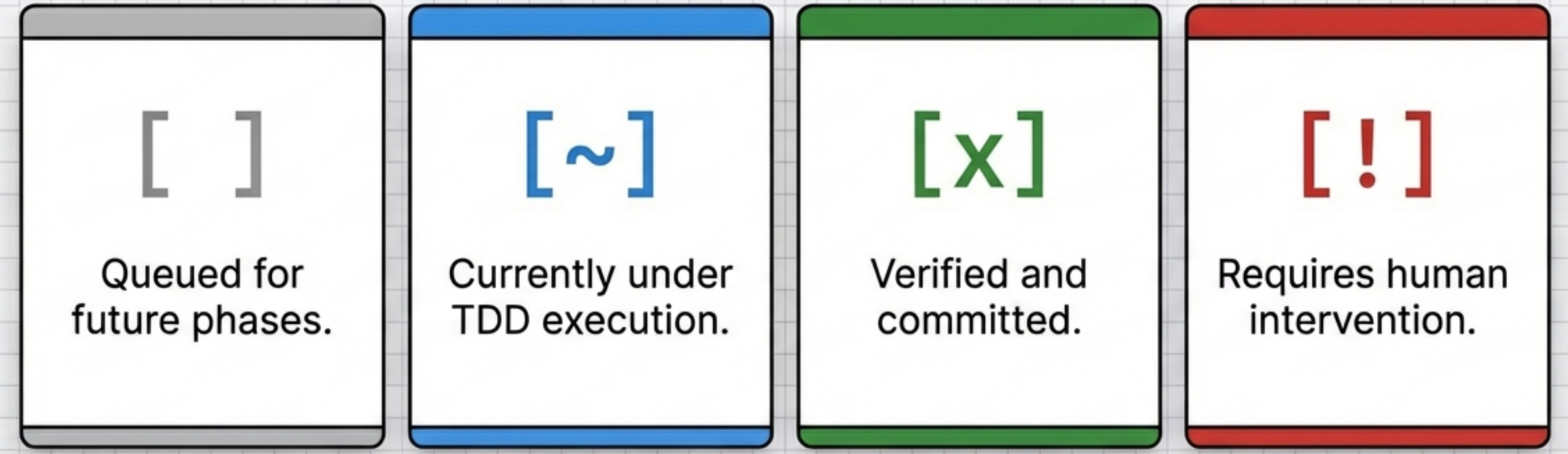
Implement minimum code to pass.

## REFACTOR

Clean up while tests remain green.

# Managing State & Visibility

`> /draft:status`

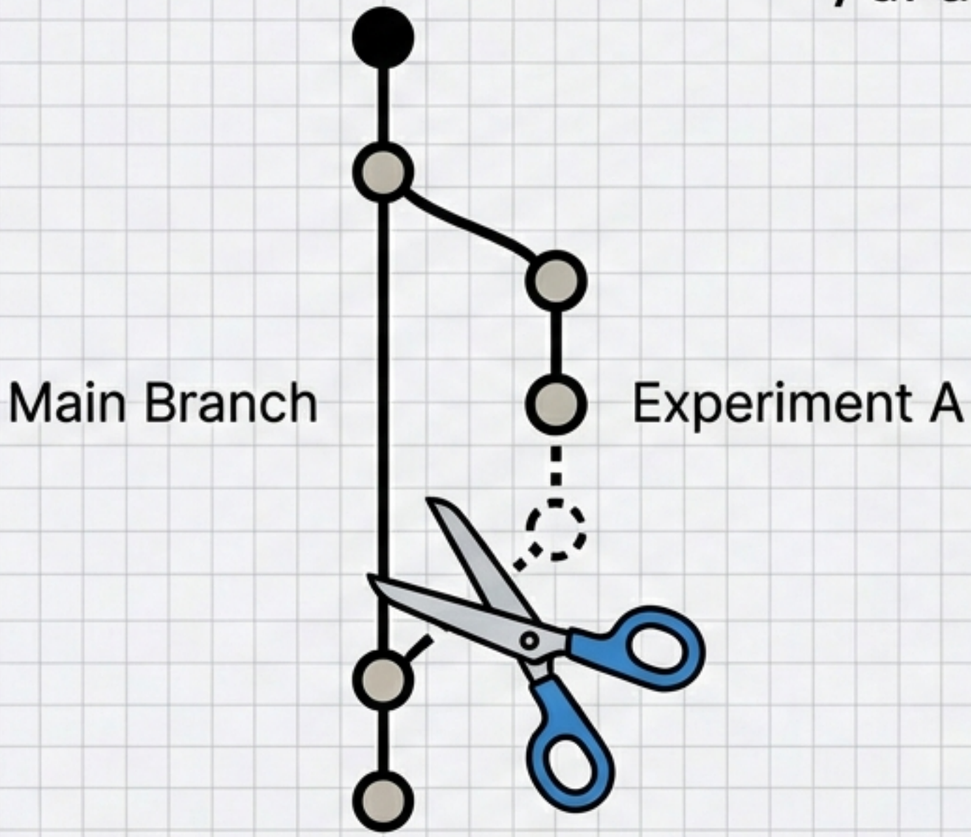



State is tracked automatically in `metadata.jon` providing an instant snapshot of project health.

# Safety & Reversibility

Context shouldn’t become clutter.

```
> /draft:revert
```

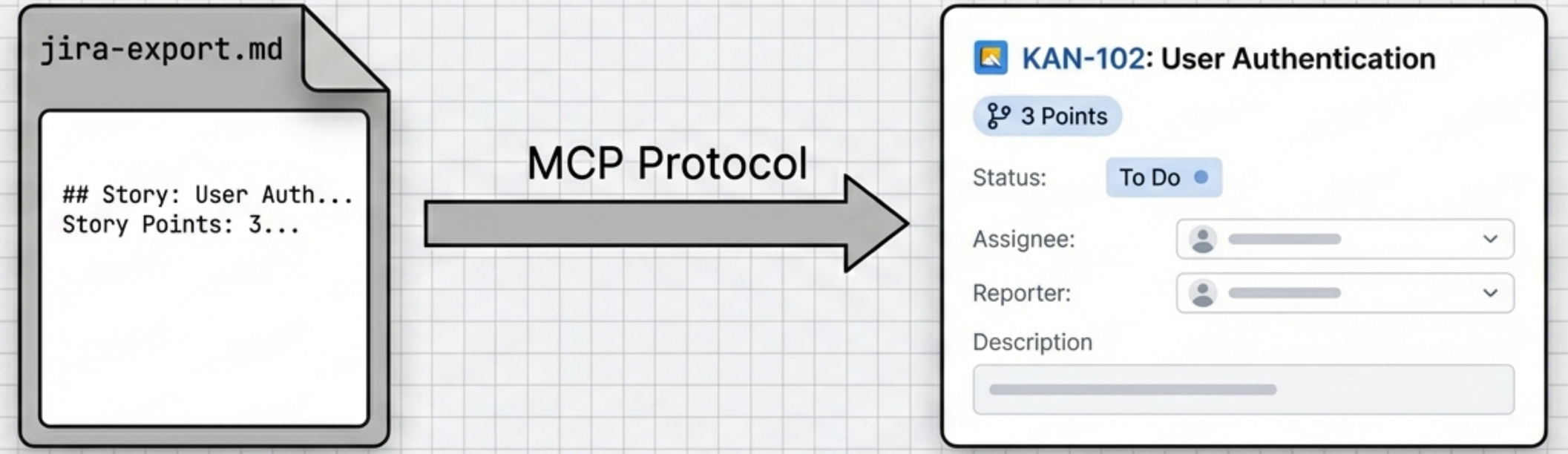



## Git-Aware Rollback.

Mistakes happen. The revert command allows you to cleanly rollback a specific task, an entire phase, or a whole track without polluting the project history.

# Bridging the Enterprise Gap

From Draft Context to Jira Tickets.




- **1. Preview:** Generate export file.
- **2. Review:** Verify story points (auto-calculated based on task count).
- **3. Create:** Push to Jira. Inter Regular.

# The Tooling Ecosystem

The methodology relies on standard markdown files, making it platform-agnostic.

## Claude Code

```
claude --plugin-dir
~/.claude/plugins/draft
```

Native Plugin Experience

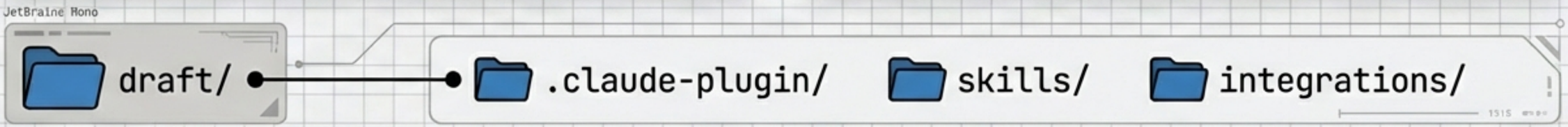
## Cursor

```
.cursorrules
```

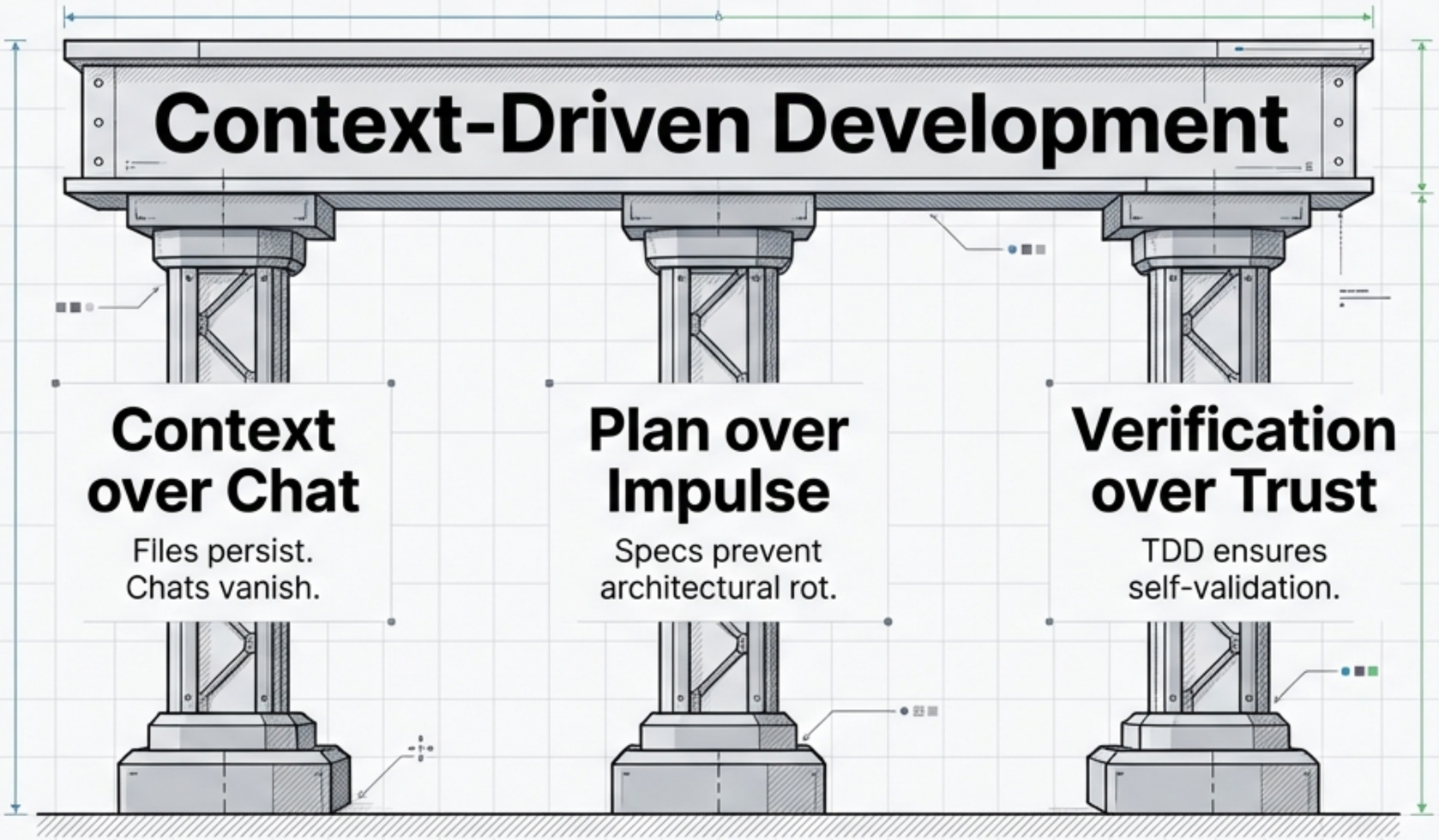
```
@draft implement
```

Rule-based Integration

JetBrains Mono



# Structured AI Engineering




# Measure twice. Code once.