David Vydra

Builder of Apps and Tools

Development Expert, SAP

Email

david@vydra.net

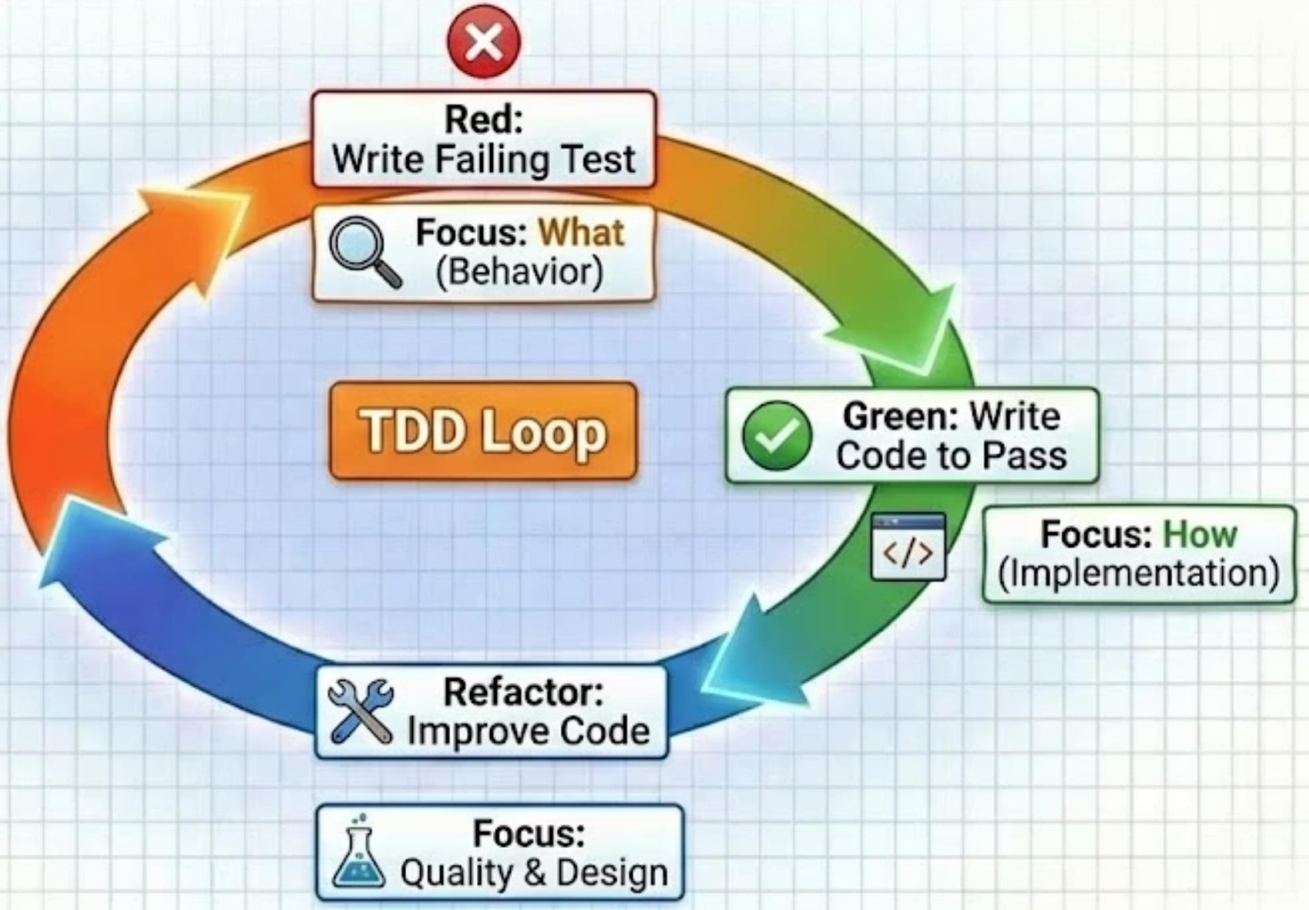
linkedin.com/in/dvydra

Website

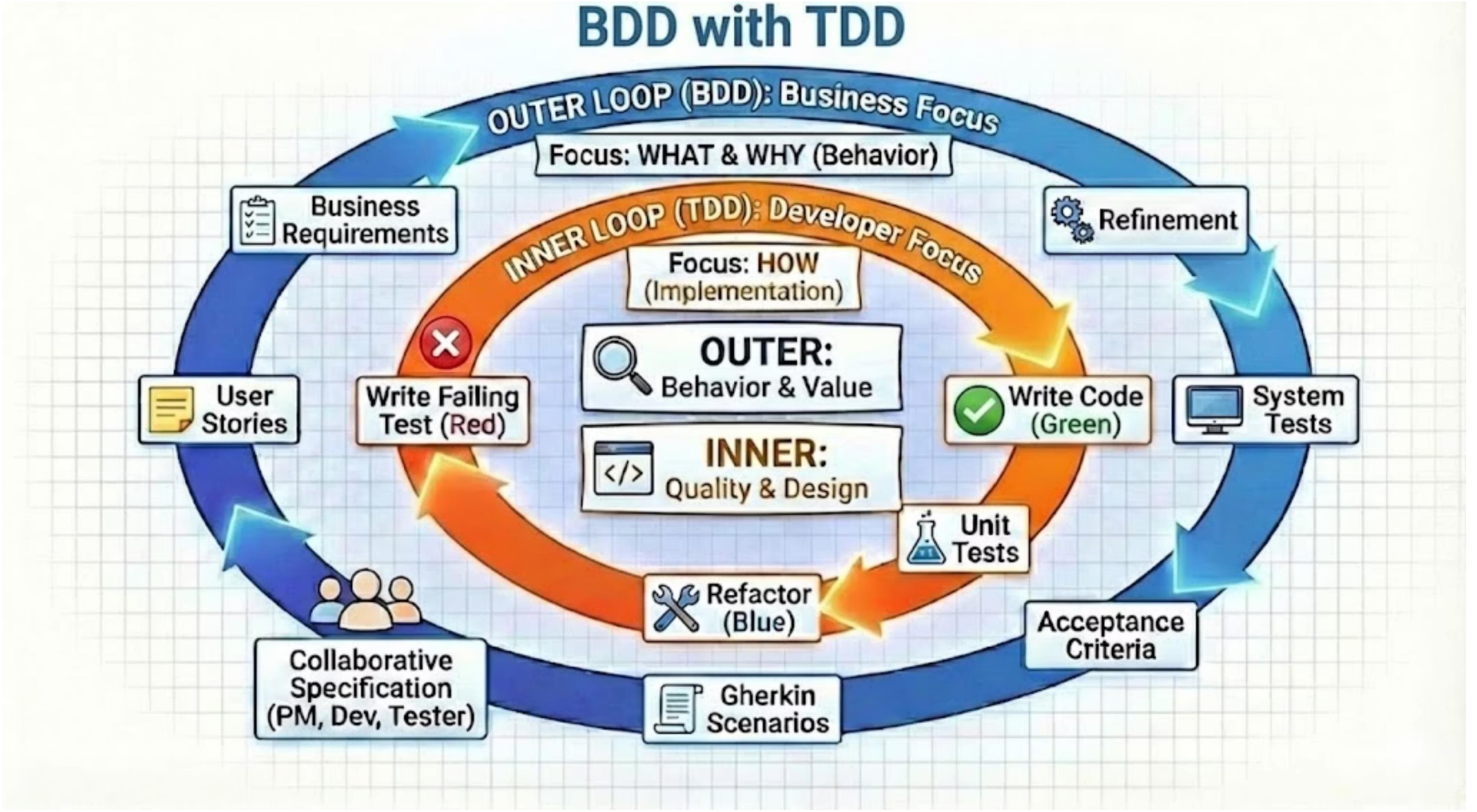
specdriven.app

A Bit of History: Last 25+ Years

TDD Cycle: Red, Green, Refactor



BDD with TDD


OUTER LOOP (BDD): Business Focus
Focus: WHAT & WHY (Behavior)
INNER LOOP (TDD): Developer Focus
Focus: HOW (Implementation)
OUTER: Behavior & Value
INNER: Quality & Design
Business Requirements
Refinement
User Stories
Write Failing Test (Red)
Write Code (Green)
System Tests
Unit Tests
Refactor (Blue)
Acceptance Criteria
Collaborative Specification (PM, Dev, Tester)
Gherkin Scenarios

BDD Explained

Behaviour-Driven Development: From TDD to Collaborative Specification

Evolution from TDD

Emerging to improve TDD, focusing on behavior and shared understanding.

The Triad Collaboration

Brings together the "Three Amigos" to define executable specifications collaboratively.

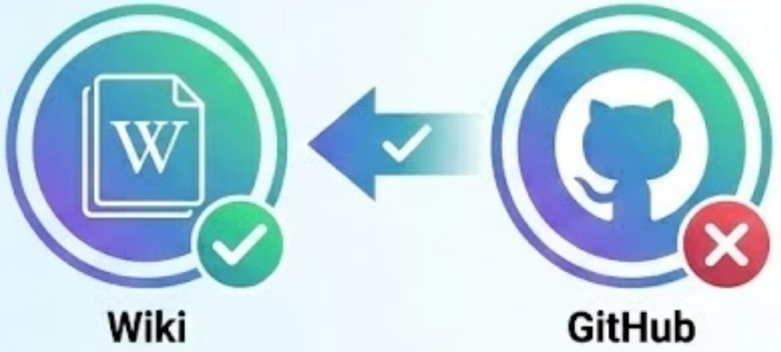
Discovery Workshop & Refinement

Key practice to refine specifications from original PRD or user story, focusing on examples.

Focuses on the 'what' and 'why' before the 'how' | Source: Modern Software Engineering Practices

How to BDD?

Wiki vs. GitHub for BDD Practice



We found that the Wiki is a much more productive place to practice BDD than directly in Github.

Cucumber: The Worst of Both Worlds

Focuses on the 'what' and 'why' before the 'how' | Source: Modern Software Engineering Practices

Search

Navigation

Aski pages

Wiki

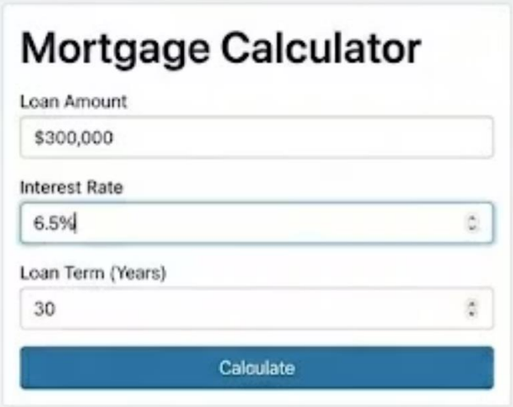
Mortgage Calculator

BDD Scenarios

Mortgage Calculator

Wiki: Mortgage Calculator - BDD Scenarios

Feature: Mortgage calculator



Scenario: 30-year mortgage payment schedule

Given a user has entered a loan amount of $300,000, an interest rate of 6.5%, and a loan term of 30 years

When the user clicks the 'Calculate Schedule' button

Then a monthly payment schedule table should be displayed with 360 rows showing payment date, principal, interest, and remaining balance for each month.

```java
public class MortgageCalculatorIT {

    //<wiki url>/BDD/MortgageCalculator-BDD-Scenarios
    @Test
    void shouldCalculate30YearMortgagePaymentSchedule() {
        // Given
        double principal = 300000.00;
        double annualInterestRate = 6.5;
        int loanTermYears = 30;
        int totalPayments = loanTermYears * 12;

        // When
        double monthlyRate = annualInterestRate / 100 / 12;
        double monthlyPayment = principal * (monthlyRate * Math.pow(1 + monthlyRate, totalPayments))
                / (Math.pow(1 + monthlyRate, totalPayments) - 1);

        // Then
        assertEquals( expected: 360, totalPayments, message: "30-year mortgage should have 360 monthly payments");
        assertTrue( condition: monthlyPayment > 0, message: "Monthly payment should be positive");
        assertEquals( expected: 1896.20, monthlyPayment, delta: 0.01,
                message: "Monthly payment should be approximately $1,896.20");
```

Executable Specifications for your components

- To a large degree your success will depend on your testing strategy.
- Well-written component (integration) tests that are traceable to the BDD specs makes all the difference.

Testdriven 2.0 ?

David Vydra Follow 2 min read · Jun 29, 2024

6 2

Twenty five years ago I was fortunate to be offered an opportunity to review the draft of Extreme Programming by Kent Beck as part of the Silicon Valley Patterns reading group. This was the most impactful event in my career. I quickly ported JUnit to the specialized Forte 4GL language I was using at the time and thus began my love affair with test driven development. I started the testdriven.com site to promote TDD and was very involved in the community for some time. As with any technique that requires significant skill and judgement, TDD was abused and most developers never learned how to do it effectively. Today, developer-written automated tests are ubiquitous, but I find TDD a path less traveled.

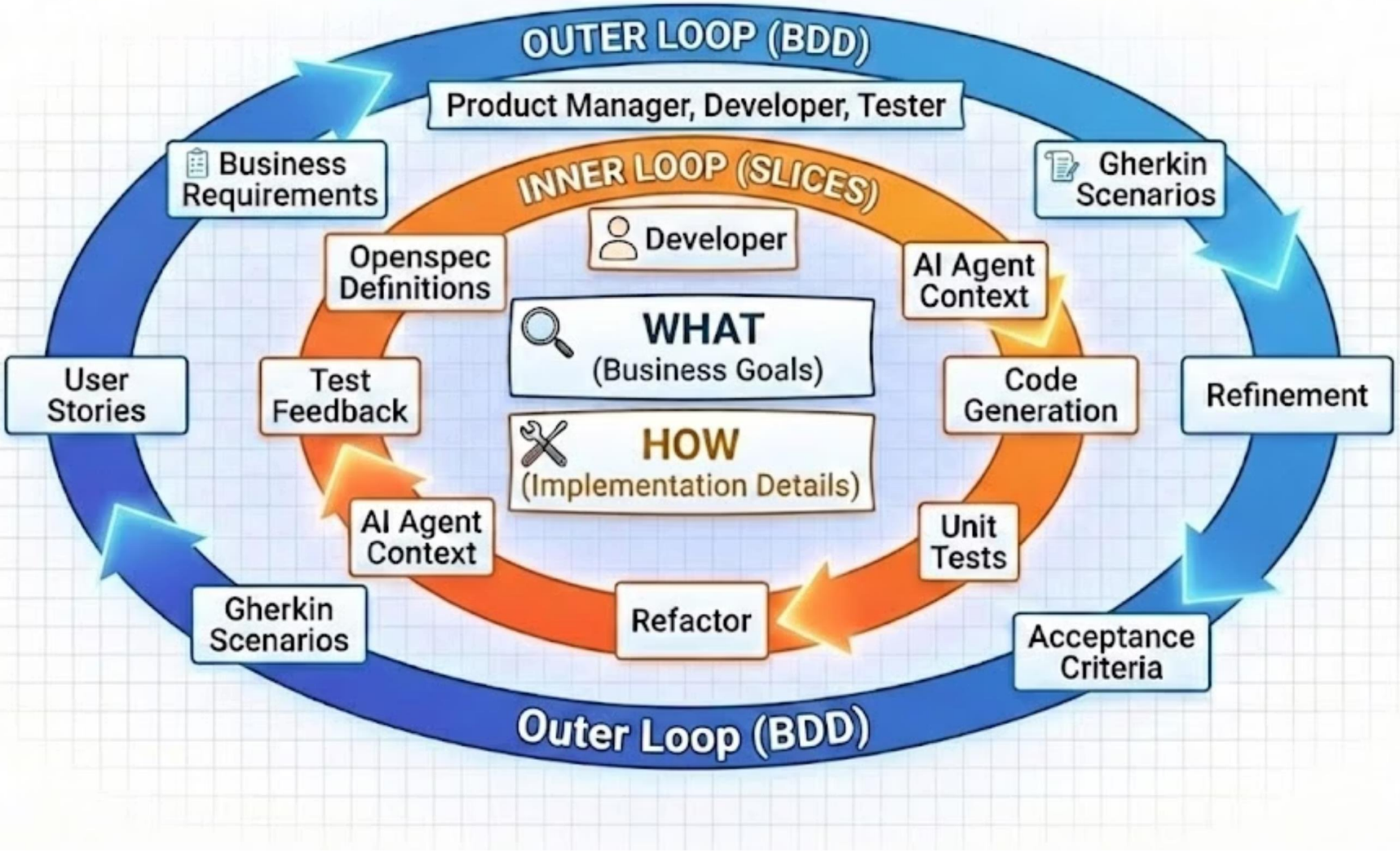
Create AI coding agent!

Supports true TDD.

Only see the code if all tests passed

Lessons Learned: need specs in English too slow for classic TDD experience

Spec-driven Development



Why work in Slices?

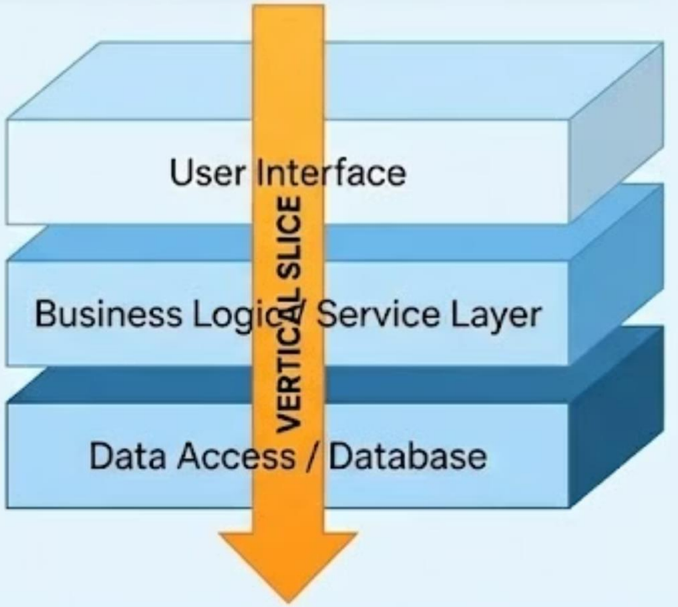
- Most developers have always worked this way, TDD never became mainstream.
- Speed: LLMs need research and thinking time to reduce Hallucinations.
- Planning is good for both human and AI.

What is a Slice?

A **“Slice”** (or **Vertical Slice**) is a **small, complete increment of deliverable value that cuts through all architectural layers of the system** (e.g., UI, Business Logic, Database).

Examples:

- **A feature:** A new capability that delivers end-to-end functionality for the user.
- **A scenario:** A specific user journey or interaction flow within the system.
- **Bug/Defect:** A complete fix for a reported issue, verified across all layers.
- **Refactoring:** Improving internal code structure incrementally without changing external behavior.



Slices focus on delivering tangible progress and value **in small, manageable pieces.**

Introducing OpenSpec

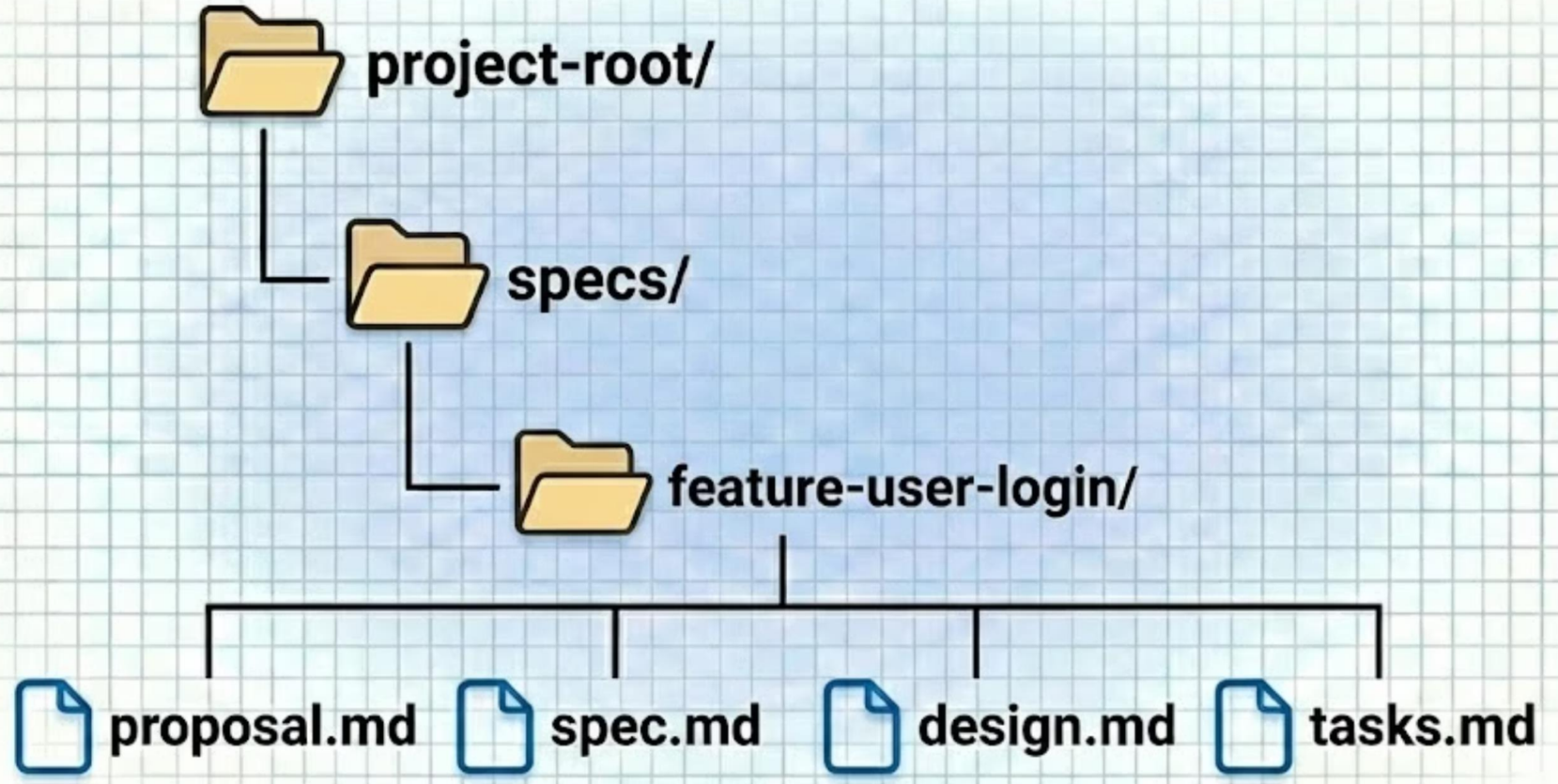
Mission

- Transform AI coding from unpredictable prototypes to structured and maintainable code
- Establish a single source of truth for intent
- Reduce halluctnations and ensure reviewable records
- Ernpower developers with spec-driven design

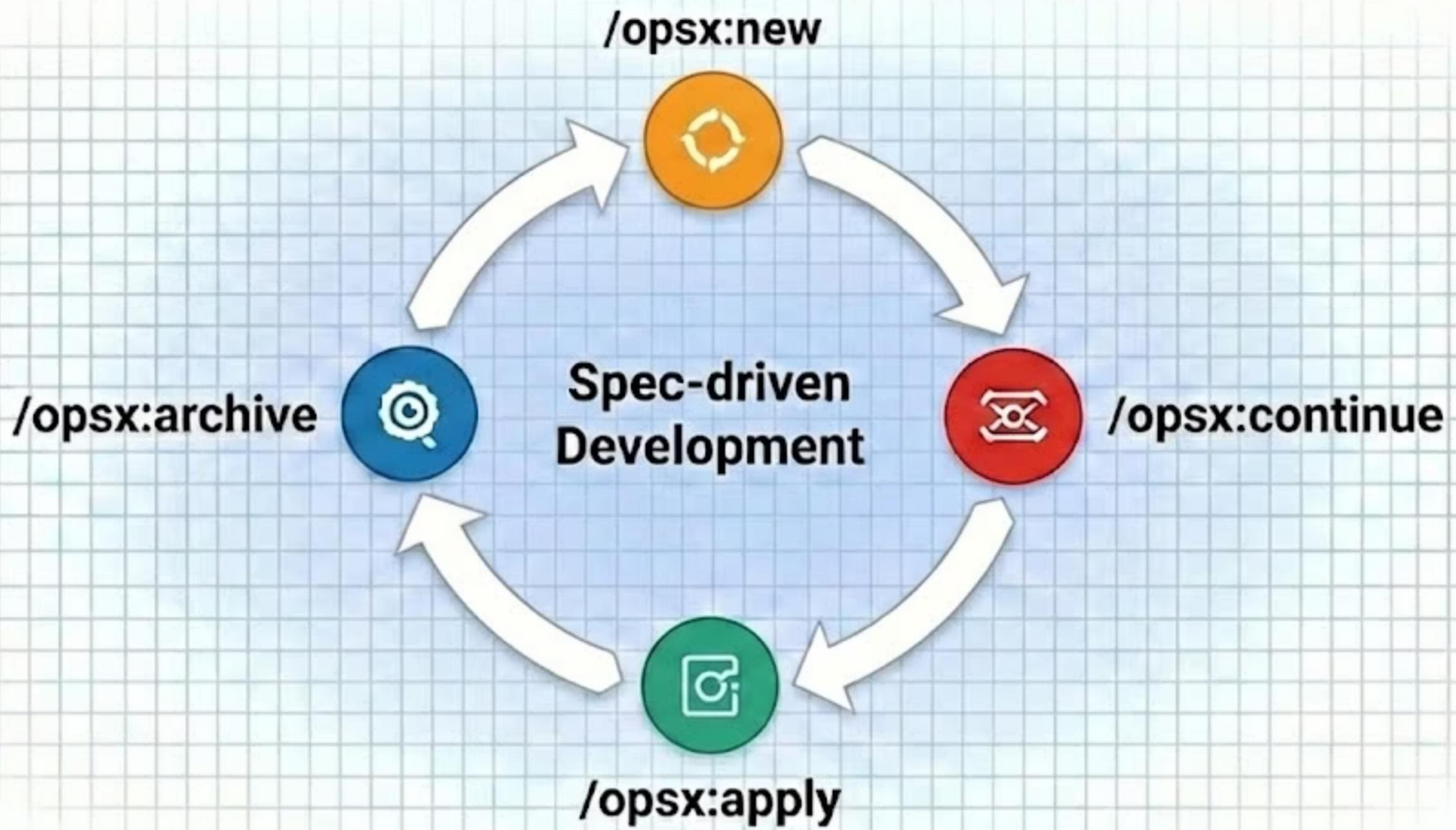
Approach: Spee Driven & Iterative Workflow

- Agree on specifications before code generation
- Utillze lightweight, Markdown-based artlfacts (Proposal, Specs, Tasks, Design)
- Embrace fluid, Iterative, not waterfall, processes
- Supports brownfield development with delta specs
- Integrates seamlessly with existing AI tools

OpenSpec Directory Structure



Introducing OpenSpec



/opsx:new Create OpenApiClient that uses auth in .jwt_token file.

Proposal.md

```
## Why

OpenAPI integration tests currently duplicate boilerplate code for reading the `.jwt_token` file,
constructing authenticated HTTP requests, and handling responses. This makes tests verbose and harder to maintain.
A reusable client class will simplify test authoring and ensure consistent authentication handling.

## What Changes

- Add new `OpenApiClient` utility class for making authenticated OpenAPI requests
- Client reads bearer token from `.jwt_token` file automatically
- Client accepts URL suffix (relative path) and constructs full URL from base config
- Client returns JSON response for easy assertions
- Client sets required headers (Authorization, sbnc-org-id, Accept)

## Capabilities

### New Capabilities
- `openapi-client`: HTTP client utility for making authenticated OpenAPI requests in integration tests.
- Handles token loading, header construction, and JSON response parsing.

### Modified Capabilities
<!-- No existing spec-level requirements are changing -->
```

Spec.md

```
## ADDED Requirements

### Requirement: Client instantiation with base URL
The system SHALL provide a constructor that accepts a base URL string. The base URL SHALL be used as the prefix
for all subsequent requests.

#### Scenario: Create client with base URL
- **WHEN** `new OpenApiClient("https://api.example.com")` is called
- **THEN** the client is instantiated and ready for requests

### Requirement: Bearer token loading from file
The system SHALL read the bearer token from the `.jwt_token` file in the current working directory.
The token SHALL be loaded lazily on the first request.

#### Scenario: Token file exists
- **WHEN** a request is made and `.jwt_token` file exists with content "abc123"
- **THEN** the Authorization header is set to "Bearer abc123"

#### Scenario: Token file does not exist
- **WHEN** a request is made and `.jwt_token` file does not exist
- **THEN** the system throws an exception with a descriptive message

#### Scenario: Token file is empty
- **WHEN** a request is made and `.jwt_token` file exists but is empty
- **THEN** the system throws an exception with a descriptive message
```

Design.md

```
## Context

Integration tests for OpenAPI endpoints require authenticated HTTP requests with:
1. Bearer token from `.jwt_token` file (3-Legged OAuth token)
2. `org-id` header from configuration (`oid`)
3. `Accept: application/json` header

Currently, each test class duplicates the token loading, header construction, and request building code.
The existing `OpenApiHeaderTest` demonstrates this pattern with ~40 lines of setup boilerplate
before any actual test logic.

## Goals / Non-Goals

**Goals:**
- Provide a simple API: `client.get("/path/to/endpoint")` returns JSON response
- Automatically load bearer token from `.jwt_token` file
- Automatically set required headers (Authorization, org-id, Accept)
- Use existing Config class for configuration values
- Support GET requests (primary use case for OpenAPI tests)

**Non-Goals:**
- Full HTTP client abstraction (POST, PUT, DELETE methods) - can be added later if needed
- Token refresh or OAuth flow handling - token acquisition is separate concern
- Caching or connection pooling optimization
- Support for non-JSON responses
```

Tasks.md

```
## 1. Core Implementation

- [x] 1.1 Create `OpenApiClient` class in `com.example.openapi` package
- [x] 1.2 Add constructor that accepts base URL string and stores it
- [x] 1.3 Add private method to load bearer token from `.jwt_token` file with lazy loading
- [x] 1.4 Add private method to get `sbnc-org-id` from Config (`oid` property)
- [x] 1.5 Implement `get(String urlSuffix)` method that makes HTTP GET request with Unirest

## 2. Header Configuration

- [x] 2.1 Set `Authorization: Bearer <token>` header from loaded token
- [x] 2.2 Set `org-id` header from Config property
- [x] 2.3 Set `Accept: application/json` header

## 3. Error Handling

- [x] 3.1 Throw descriptive exception when `.jwt_token` file does not exist
- [x] 3.2 Throw descriptive exception when `.jwt_token` file is empty
- [x] 3.3 Throw descriptive exception when `oid` config property is missing

## 4. Testing

- [x] 4.1 Write unit test for successful GET request with all headers
- [x] 4.2 Write unit test for missing token file exception
- [x] 4.3 Write unit test for empty token file exception
```

Archive of "slices"

2026-01-09-add-posthog-analytics	Add optional anonymous usage statistics (#468)
2026-01-09-fix-codebuddy-frontmatter-fields	feat: change the frontmatter of the Codebuddy Slash Commands (#462)
2026-01-15-add-nix-ci-validation	feat: add nix flake support (sorry for this duplicate) (#459)
2026-01-30-opencode-command-references	fix(opencode): transform command references from colon to hyphen form.
2026-02-17-add-feedback-command	Bulk archive completed changes and normalize source specs (#716)
2026-02-17-add-opsx-onboard-skill	Bulk archive completed changes and normalize source specs (#716)
2026-02-17-add-verify-skill	Bulk archive completed changes and normalize source specs (#716)
2026-02-17-merge-init-experimental	Bulk archive completed changes and normalize source specs (#716)
2026-02-17-multi-provider-skill-generation	Bulk archive completed changes and normalize source specs (#716)
2026-02-17-project-config	Bulk archive completed changes and normalize source specs (#716)
2026-02-17-project-local-schemas	Bulk archive completed changes and normalize source specs (#716)
2026-02-17-schema-management-cli	Bulk archive completed changes and normalize source specs (#716)

OpenSpec / openspec / **specs** /

docs-agent-instructions	feat: simplify skill installation with profiles and smart defaults in...
global-config	Add optional anonymous usage statistics (#468)
instruction-loader	feat: add instruction loader for template loading and change context (#...
legacy-cleanup	Bulk archive completed changes and normalize source specs (#716)
openspec-conventions	feat: simplify skill installation with profiles and smart defaults in...
opsx-archive-skill	Bulk archive completed changes and normalize source specs (#716)
opsx-onboard-skill	Bulk archive completed changes and normalize source specs (#716)
opsx-verify-skill	Bulk archive completed changes and normalize source specs (#716)
rules-injection	Bulk archive completed changes and normalize source specs (#716)
schema-fork-command	Bulk archive completed changes and normalize source specs (#716)
schema-init-command	Bulk archive completed changes and normalize source specs (#716)
schema-resolution	Bulk archive completed changes and normalize source specs (#716)
schema-validate-command	Bulk archive completed changes and normalize source specs (#716)

DORA AI Research Insight

Are you ready for AI-augmented Development?

"Which organizational reality do you see?"

Ready for Amplification

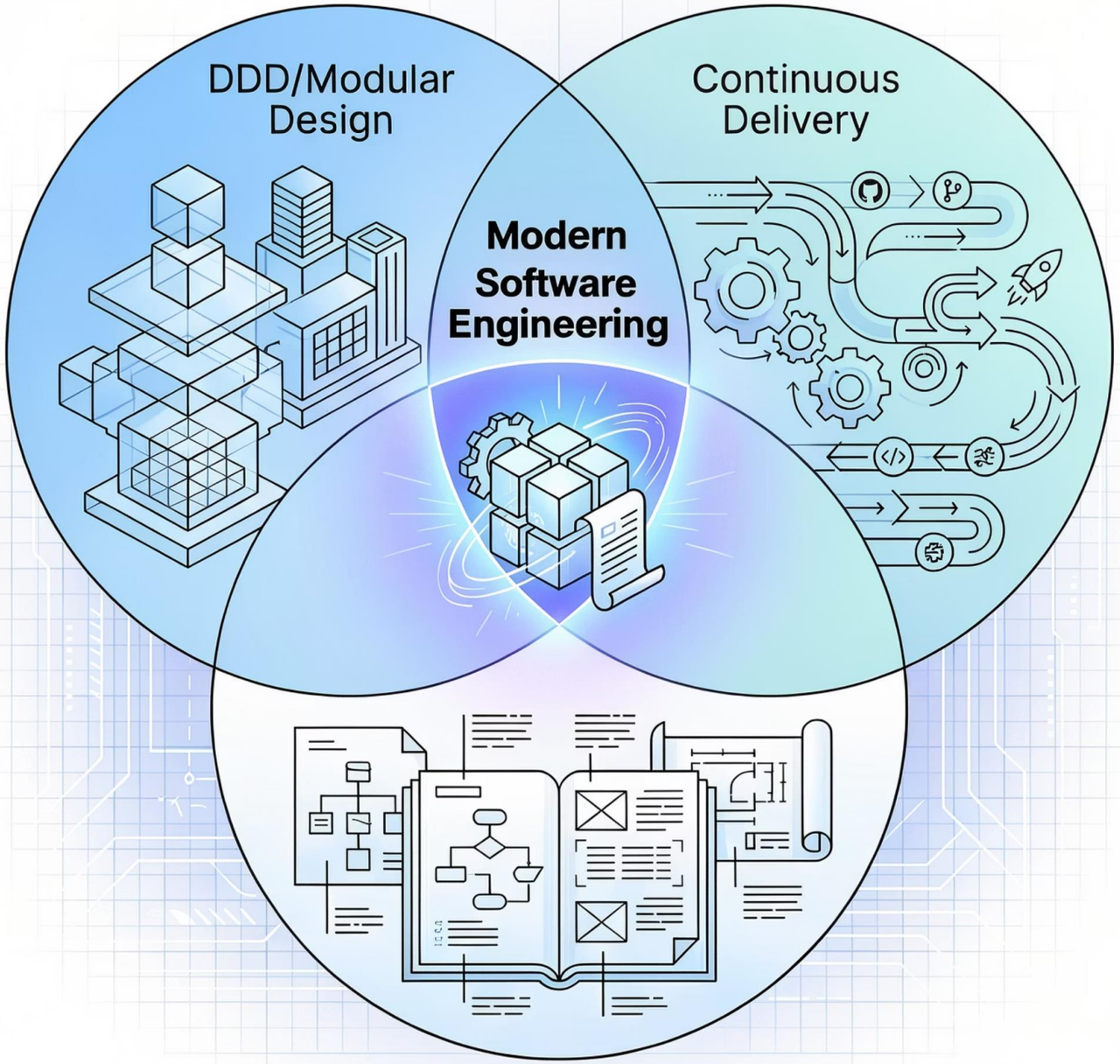
Strong Foundation:
Clear goals, efficient processes, collaborative culture. AI will accelerate success.

At Risk of Chaos

Fragile State:
Bottlenecks, instability, cultural friction. AI will exacerbate problems.

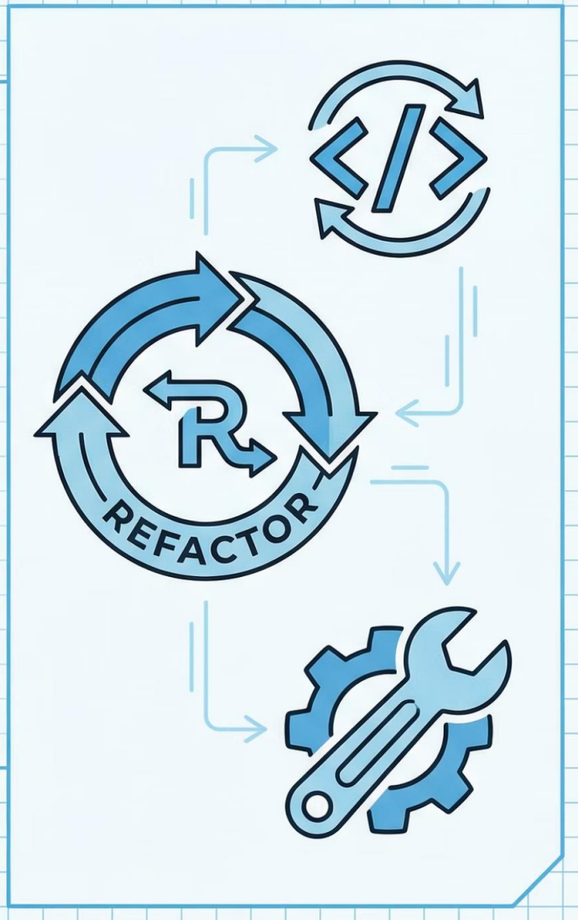
Conclusion: Assess and strengthen your organizational health BEFORE adopting AI tools.

Source: Based on DORA 2025 State of AI-assisted Software Development Report


DDD/Modular Design
Continuous Delivery
Modern Software Engineering
Spec-driven Development

Evolutionary Design Skills are key to maintainable code

- Refactoring is as important as ever!
- Your design quality can deteriorate 10x faster without constant refactoring.



What about 10x ?

Yes, and ...

More time for design

More time for testing

It was never about the code!

How do I prepare for the AI future?