



Test Data Driven Development (TDDD)

Reducing risk through pre-development modelling

Rick Spencer | Engineering Team Leader

The testing work moved. It did not vanish.

STEP 01

Testing after build

A specialist discovers edge cases after implementation.

- Traditional reactive QA model
- Edge cases found late in cycle
- High rework cost

STEP 02

Collective Specification

The Squad defines the boundaries and expected outputs upfront.

- Shift-left specification
- Upfront boundary definition
- Explicit domain expectations

STEP 03

S3

Encoded Methodology

A mandatory workflow concern ensuring data definition precedes implementation.

- Automated pipeline enforcement
- Data model as primary artifact
- Continuous verification build

We did not eliminate testing expertise; we changed where it lives in the delivery pipeline.

Logic-only verification misses data-state risk

CODE PATH VERIFICATION

Deterministic Logic

Feature behaviour is mapped by predictable code paths.

Code Path A
Inputs & Rules



Output B
Expected Result

ANALYTICS TESTING

Static Logic, Variable Data

Complexity arises from object relationships and field ranges.

Object Schema
Relational Graph

Field Ranges
Data State Variations

**Combinatorial
Data Matrix**

Data-State Risk
Hidden Failures

Without modelling, developers are forced to guess the data shape. The pipeline can run successfully, but **if the data combinations are wrong, the dashboard is wrong.**



"Test-driven development is a way of managing fear during programming." — **Kent Beck**

Same inversion, different first artifact

TDD

01. First Artifact

Write the failing assertion first

02. Behaviour Specification

The assertion defines correct behaviour

03. Implementation Loop

Red → green drives implementation

04. Long-Term Value

The test remains as regression

TDDD

01. First Artifact

Write the expected data model first

02. Behaviour Specification

The input/output model defines correct behaviour

03. Implementation Loop

Actual-vs-model comparison drives implementation

04. Long-Term Value

The same rows remain as regression

CORE PHILOSOPHY

Build the world first,
then see if the software can describe it.

The Persistent Data Safety Net

STEP 01

Requirement

Define core objective & criteria

STEP 02

Data Model

(Spreadsheet)

Author expected input/output

STEP 03

Feature Build

Develop software logic

STEP 04

Expected vs. Actual

Compare result with oracle

STEP 05

Regression Assertion

Lock scenario into safety net

CORE RULE

Fix the model first, then the code.

A discovered bug permanently widens the safety net; it becomes a new scenario, not just a patch.





An Agreed Expected Result

ID Scenario Input State Expected Output

S1 Standard allocation baseline 20 hours scheduled 0 hours variance

S2 Partial week allocation 15 hours scheduled 0 hours variance



S3 Resource reaches weekly maximum CRITICAL SCENARIO BOUNDARY 40 hours scheduled 0 hours variance

S4 Over-allocation threshold test 41 hours scheduled +1 hr variance (Flagged)

S5 Max capacity extended rule 50 hours scheduled +10 hrs variance (Hard Lock)

S6 Zero allocation baseline 0 hours scheduled 0 hours variance

S7 Multi-project split allocation 40 hours total (20/20) 0 hours variance

Core Rule

PM and engineering collaboratively author expected results before writing code.

Key Takeaway

First lock down the shared expectation model; then write the software code to satisfy it.

The model becomes the comparison oracle

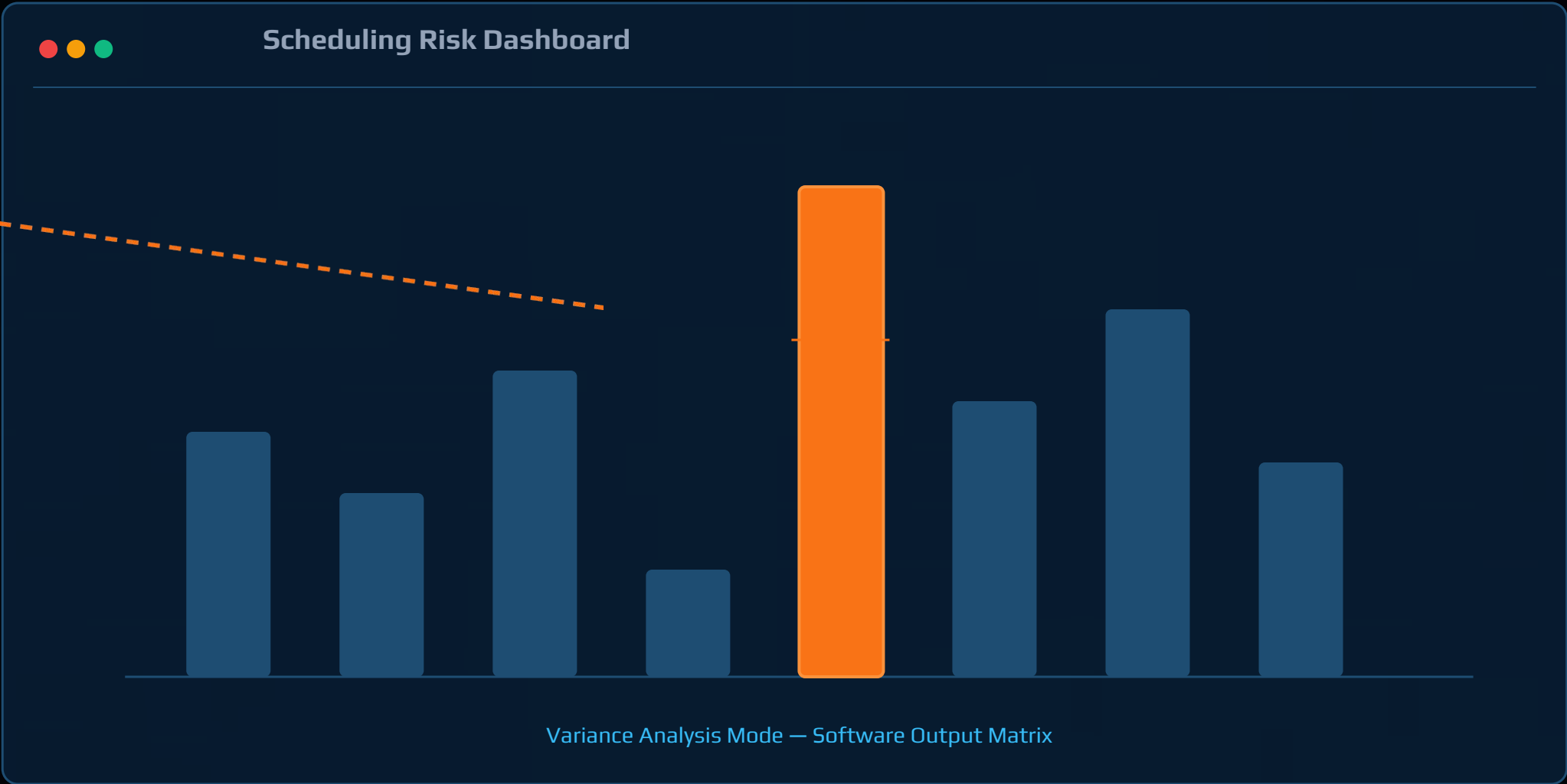
TARGET SCENARIO

Scenario S3

40 hours scheduled

0 hours variance

The expected input/output boundary defined in the data model.



AUTOMATED VERIFICATION

The software output is compared with the expected result for the modelled scenario.

The S3 scenario now locks in as the automated regression test.

Validating the Bubble

IN-SCOPE VERIFICATION

Validates

Scenario Conformance

Conformance to stated scenarios within the exercised environments.

Model Representation

The important behaviour represented by its model.

OUT-OF-SCOPE LIMITS

Does Not Prove

1. Model Completeness

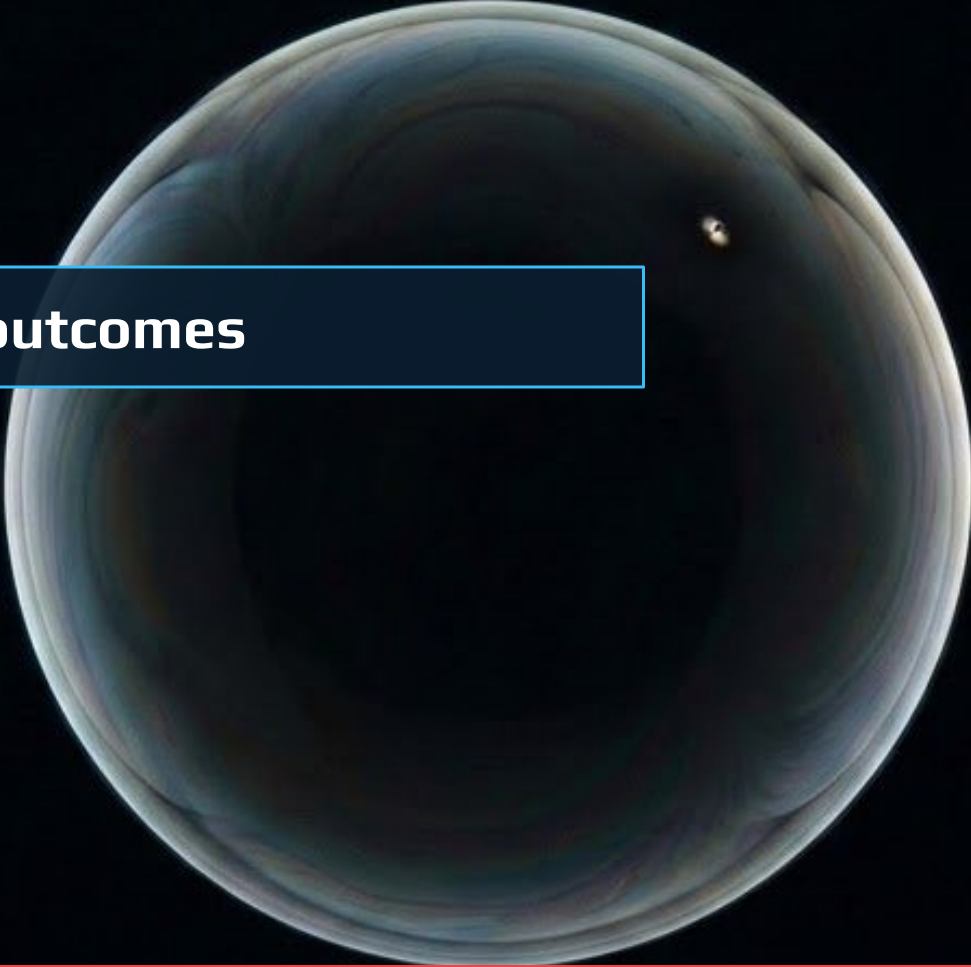
That the model is complete.

2. Oracle Accuracy

That the test oracle is correct.

3. Pipeline Integrity

That surrounding pipeline orchestration is working.



One requirement, two outcomes

Zero-hours filter scenario

- Data model completed: 4 Sep
- Feature completed: 8 Sep

Delivered against the modelled scenario

Max hours cap

Named in the requirement, but never modelled.

Missing scenario: "Assignments that generate a future prediction that takes the resource over 40 hours a week."

The control was followed: the data story existed and came first.

It covered one feature and omitted the other.

AI reduces the cost of building the proof

THE ACCELERATOR

AI-Assisted Workflow

memory

STEP 01

Data Creation

auto_awesome

STEP 02

Test Generation

arrow of workflow

check_circle AI reduces the historical cost of creating massive datasets.

check_circle Enforces workflow sequencing at refinement and readiness.

THE JUDGEMENT

Human Governance

rule

STEP 03

Scenario Selection

verified_user

STEP 04

Approval

arrow of workflow

gavel AI can suggest scenarios, but it cannot guarantee the model describes the right world.

gavel Deciding what matters remains a human judgement.

Where TDDD thrives and where testing expertise shifts

HIGH VALUE

Where it Shines

`check_circle` Known starting state
`check_circle` Defined action or trigger
`check_circle` Observable outcome
`check_circle` Repeatable setup

LIMITATIONS

Where it Struggles

`remove_circle_outline` Subjective UX
`remove_circle_outline` Exploratory discovery

STRATEGIC SHIFT

The Tester's Role

Testing expertise moves earlier: shaping scenarios, challenging boundaries, and using exploration to expose omissions.

The One-Feature Pilot

Core Team

1 PM + 1 Dev + 1 Tester

Pilot Scope

1 Data-Dependent Feature

Approach

Start small without changing governance

Traceable Relationship Matrix (6-Column Schema)

01. ID

Unique Keys

Standard identifiers for tracking scenarios across test suites.

02. Req & Scenarios

Feature Intent

Clear specification statement for the single feature under pilot as well as the scenarios to be covered

03. Starting State

Data Conditions

Pre-conditions, parameters, and environment state before execution.

04. Action or Trigger

Cause

What causes the behaviour

05. Expected Output

Deterministic Result

Exact predicted outcomes modelled prior to implementation.

06. Regression Evidence

Durable Evidence

Verifiable test logs, fixtures, and execution traces confirming pass state.

Required Scenario Modelling (5–7 Cases Before Implementation):

1. Normal • **2. Zero State** • **3. Boundary** • **4. Invalid** • **5. Historically Troublesome**

DISCIPLINE

“The spreadsheet format doesn’t matter; the traceable relationship between the scenario and the regression evidence does.”

What holds the bubble together across products?

DATA DRIFT

Product Boundaries & Fixtures

hub When data flows across product boundaries, local feature fixtures drift apart.

sync_problem Isolated test setups fail to capture cross-product interactions and schema evolutions.

verified Requires shared reference modelling and continuous boundary contract validation.

SCALE BOTTLENECK

AI Implementation & Review

code If AI is doing more of the implementation, line-by-line code review becomes a bottleneck.

speed High-velocity automated code generation overwhelms manual inspection processes.

gavel Review focus must shift from syntax checking to scenario modelling and behavioural evidence.

QUESTIONS FOR ANOTHER DAY

“What changes across product boundaries, and what evidence should people review as AI writes more implementation?”

Five questions for your first modelling session

01

Specification

What behaviour has been specified?

02

Input Data

What input data exposes it?

03

Expected Output

What expected output makes success checkable?

04

Evidence

What evidence demonstrates each scenario?

05

Boundary

What important behaviour remains outside the boundary?

"Build the world first, then see if the software can describe it."