From AI-assisted coding to governed SDTM derivation development

PHUSE SDE West Windsor
22 July 2026

Rostislav Markov

Amazon Web Services, Inc.

RESET THE QUESTION

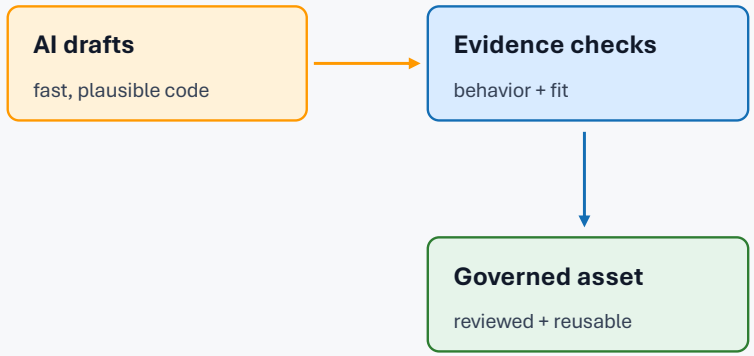
AI-assisted coding is not governed development

A model can draft code quickly.

But an SDTM derivation still must be...

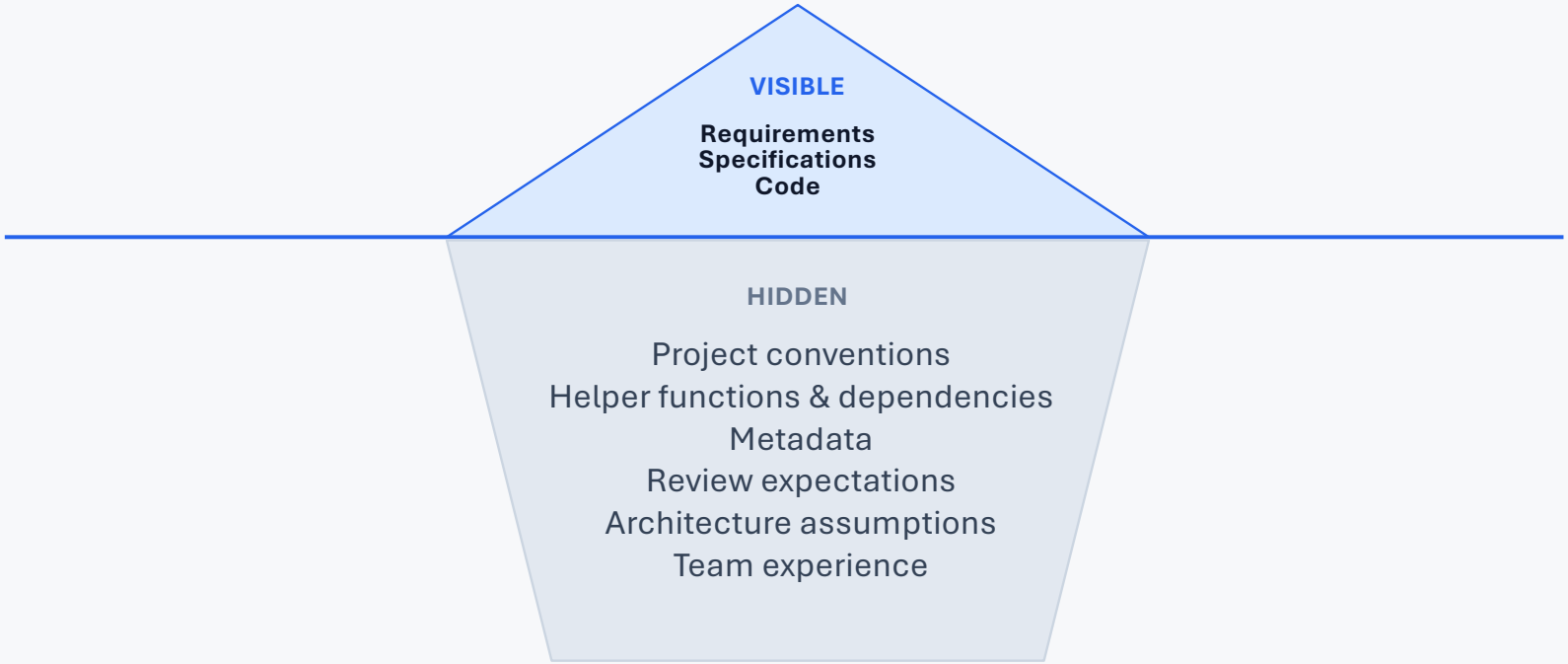
- correct
- traceable
- validated
- integrated
- reusable
- reviewable

This session is about the maturity journey:



What engineering knowledge must we make explicit?

Where engineering knowledge hides



LLMs only see what we make explicit.

EVALUATION SAMPLE

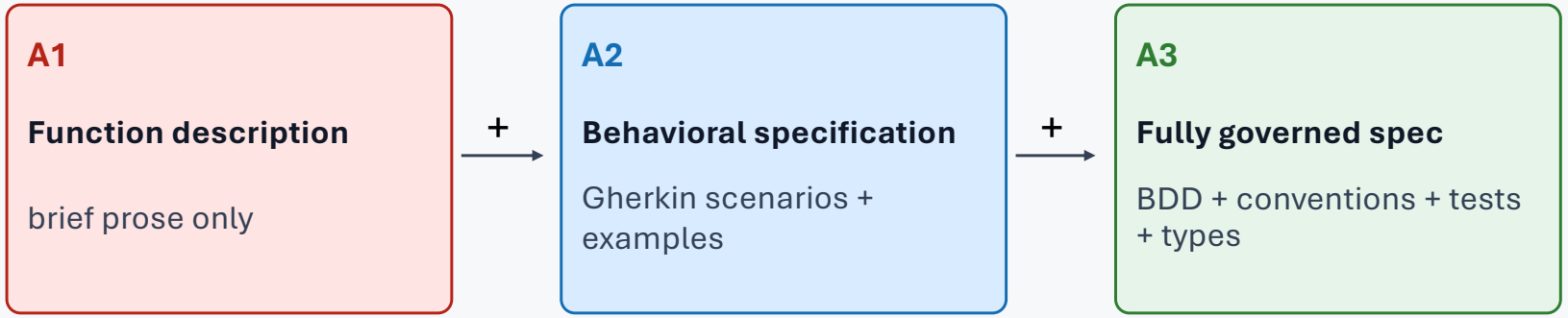
The eight derivations tested (ground truth in Python)

Function	Intended behavior	Complexity
get_first_dose	Earliest EXSTDTC → RFSTDTC/RFXSTDTC in DM	Low
get_last_dose	Latest EXENDTC → RFENDTC/RFXENDTC in DM	Low
get_last_contact	Latest *DTC across all domains → RFPENDTC in DM	Medium
call_patient_death	Death flags from DS/DD → DTHFL/DTHDTC in DM	Medium
call_subjid	SUBJID propagation from DC to target domains	Medium
call_suppqual	SUPPQUAL/SQAP generation from reference metadata	High
create_epoch_taetord	EPOCH/TAETORD derivation from SE with tie-breaking	High
create_se_domain	SE domain creation via SQL queries against source domains	High

The functions span simple repeatable patterns, multi-domain derivations, metadata-driven generation, and infrastructure-dependent logic.

DESIGN

Evaluate impact of increasing governance on AI model output



The model* and target functions stayed fixed.
The variable was the engineering context supplied to the model.

**Claude Sonnet 4.6 via Amazon Bedrock*

EVALUATION

Two outcomes: correct derivation vs usable project code

Functional correctness (F%)

Does the function solve the right problem?

- ✓ Correct algorithm
- ✓ Correct join keys
- ✓ All target vars mapped
- ✓ Validation rules & exact errors
- ✓ Edge cases; no side effects

Integration quality (I%)

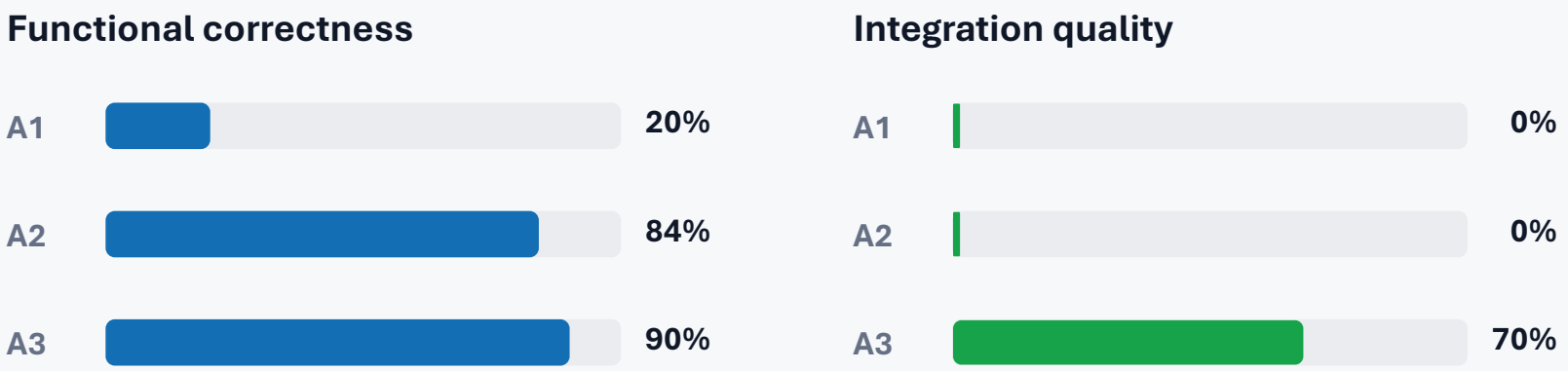
Does the function fit the project?

- ✓ Imports, decorator, configuration
- ✓ Signature and return structure
- ✓ Test harness compatibility

A function can be algorithmically correct and still fail the framework.

EVIDENCE

Result: behavior fixed semantics; conventions fixed integration



Specification quality changes the failure mode: from semantic errors (wrong algorithm) to mechanical errors (wrong import path).

EVIDENCE

The result decomposed cleanly

Behavior examples solved semantics

Input/output rows
Named target variables
Edge cases
Expected error messages

20% → 84%

Project context solved fit

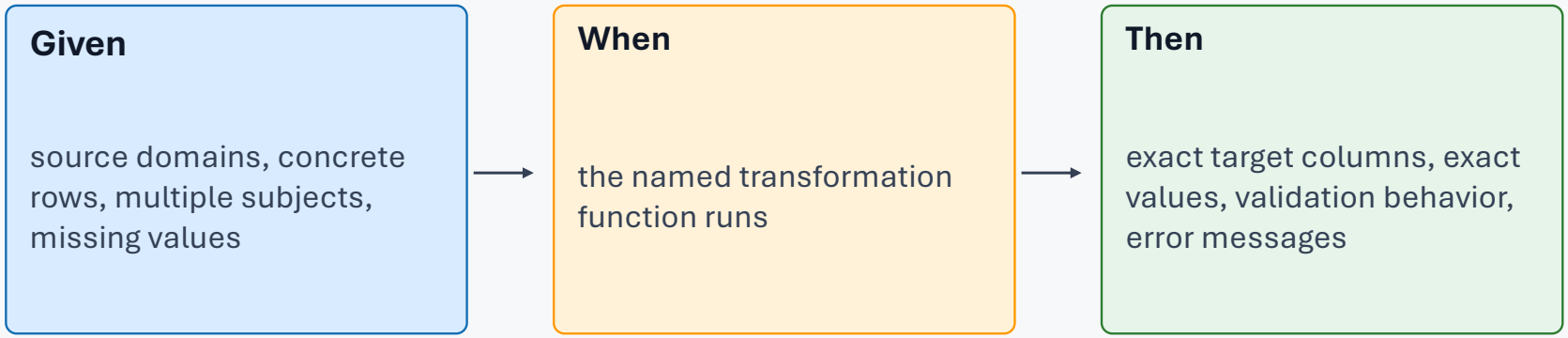
Imports
Decorators
Types
Helpers
Test harness

0% → 70%

Same model. Same target functions. Different engineering artifacts.

WHAT FIXED SEMANTICS

What behavior-driven specifications added



This explains the 20% → 84% jump:
the model stopped guessing the derivation archetype.

INTERPRETATION

Loose prompts made the model fill gaps with generic assumptions

Dose derivations

Simple functions became large treatment-period engines with extra flags.

Last contact

A maximum-date derivation became a follow-up status engine.

SUBJID propagation

The model constructed an ID instead of propagating the existing one.

If behavior is only prose, the model fills the empty space with plausible domain knowledge.

INTERPRETATION

Failure taxonomy: every error pointed to a missing artifact

Behavior

Wrong logic

Convention

Inconsistent style

Dependency

Wrong imports

Metadata

Wrong input structure

Architecture

Doesn't fit the system

Capability

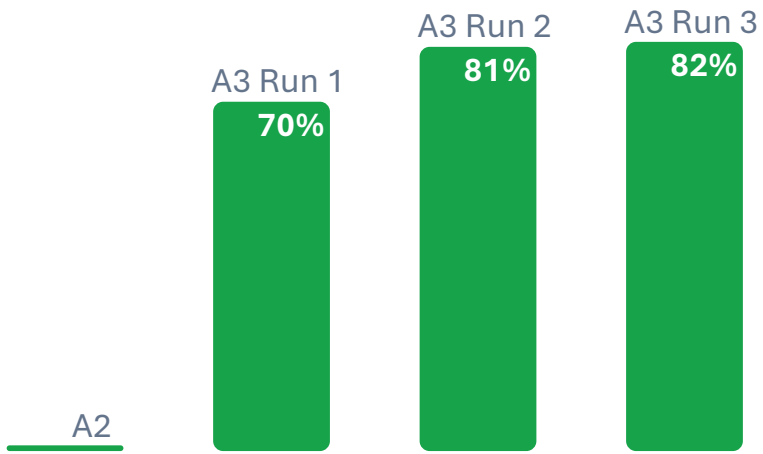
Reinvented utilities

Failures are artifact context gaps, not hallucinations.

INTERPRETATION

Iterative convention refinement improved integration

Average integration quality across runs (A3)



Addressable deviations: 19 → 7 → 6

The first conventions unlocked integration. Later revisions helped complex functions, but returns diminished as remaining gaps became more specific or irreducible. This is because supplied artifacts remained same (A3).

Conventions should not be written once.

BOUNDARY OF PROMPTING

Some gaps cannot be fixed by convention text alone

Convention-addressable

imports
types
copying
return patterns

Convention-resistant

strong model priors
archetype selection
exact wording

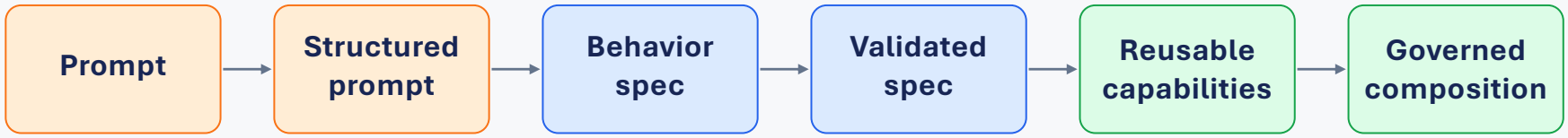
Irreducible from prompt

cross-function dependencies
reference schemas
infrastructure patterns

The long-term answer is not a bigger prompt. It is a better engineering system.

EVOLUTION OF PROMPTING

From prompt engineering to specification engineering



The direction of travel is not longer prompts.
It is durable, reviewable, reusable specifications.

FROM CODE TO DEVELOPMENT PROCESS

A governed SDTM derivation is an artifact package

Implementation

Python function

Behavior

BDD feature

Evidence

pytest suite

Metadata

inputs + parameters

Conventions

structure + helpers

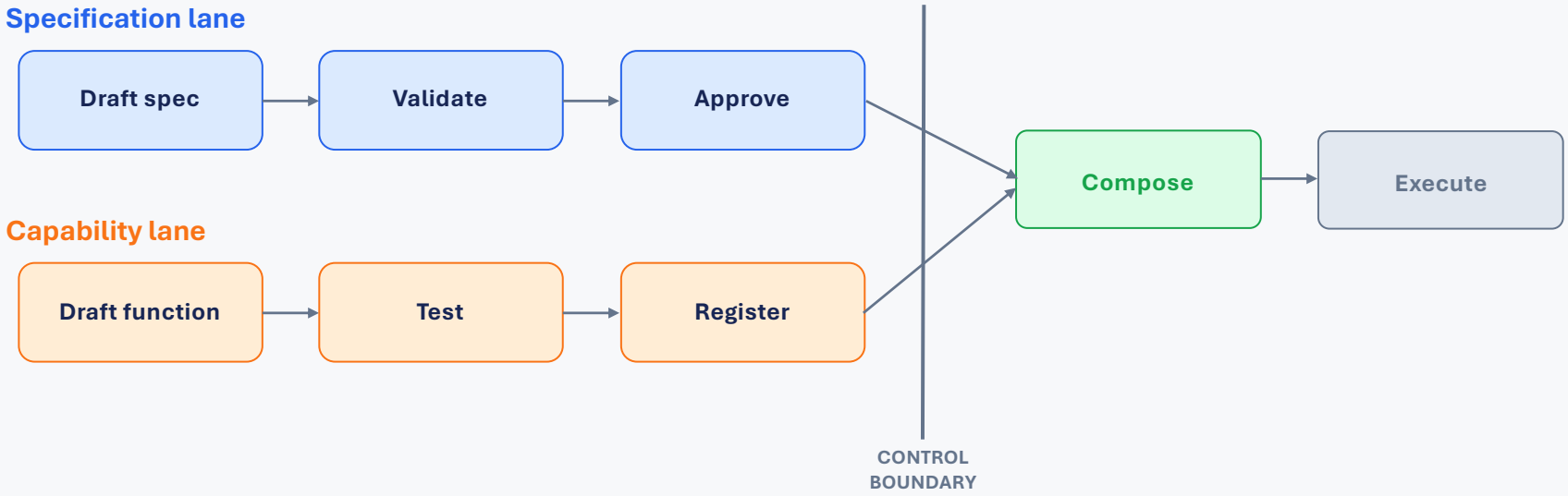
Registry

approved reusable function

AI can draft pieces of the package.
The SDLC decides whether the package becomes reusable.

GOVERNED DEVELOPMENT

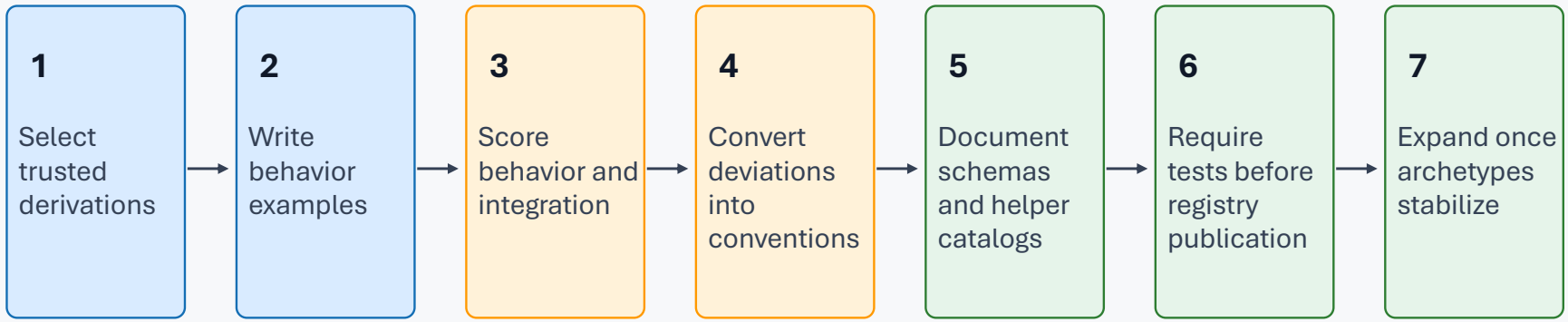
The control boundary: AI may author; approved artifacts execute



Draft artifacts are not executable.
Only approved, versioned specifications and approved capabilities cross the boundary.

ADOPTION PLAYBOOK

A practical path for a statistical programming team



Start where the model is most confused:
explicit examples, shared conventions, and executable tests.

CLOSE

What to stop doing and what to start doing

STOP

- Benchmarking models before defining behavior
- Treating prompts as throwaway text
- Reviewing generated code without executable evidence
- Letting AI infer project conventions

START

- Writing concrete examples
- Maintaining convention libraries
- Including test contracts
- Registering reusable capabilities
- Measuring correctness and integration separately

CLOSE

Three takeaways

1. **Better specifications matter more than prompt tricks.**
2. **Functional correctness and system integration are different problems.**
3. **Govern the artifacts that guide AI, not only the code it produces.**

Better models can generate better code.

Better specifications make that code dependable.

Thank you

Backup

REFERENCE

Functional correctness scorecard

Area	#	Check	Pass Criteria
Core logic (F1)	F1.1	Correct core operation	Implements the right algorithm
	F1.2	Correct join key	Links source to target on correct key(s)
	F1.3	All target variables mapped	Every target column from BDD examples is written to
	F1.4	Multi-subject support	Processes all subjects in one call
Validation (F2)	F2.1	Validates source domain	Rejects invalid source with error
	F2.2	Validates target domain	Rejects invalid target with error
	F2.3	Error messages exact match	String matches BDD-specified messages
Edge cases (F3)	F3.1	Preserves data formats	Output values match input string formats
	F3.2	Edge cases covered	All BDD scenario variants handled
	F3.3	No side effects	Does not mutate inputs or add unexpected columns

REFERENCE

Integration quality scorecard

Area	0	1	2	3
Imports, decorator & configuration (I1)	No sdtmutils	Partial imports	All imports, minor gaps	Exact match
Signature & structure (I2)	Wrong name/types	Partially correct	Correct with minor deviations	Exact match
Test pass rate (I3)	0%	1% – 49%	50% – 99%	100%

EVIDENCE

AI-generated code failed for different reasons

Derivation type	Main challenge	Most useful artifact
First/last dose	semantic over-expansion	concrete examples
Last contact	wrong clinical archetype	maximum-date examples
SUBJID propagation	model prior: construct ID	copy/join directive
SUPPQUAL	metadata-driven structure	reference schemas
EPOCH/TAETORD	tie-breaking and sequencing	edge cases + metadata
SE creation	infrastructure and SQL contracts	approved runtime pattern

FUNCTIONAL FAILURE MODES

When intent was vague, AI solved plausible but wrong clinical problems

Dose date derivations

Expected: earliest / latest EX dates into DM reference variables.

Generated: treatment-period engines with PK half-life extensions and extra flags.

Why: “dose derivation” triggered broader clinical assumptions.

Last contact

Expected: maximum *DTC across source domains into RFPENDTC.

Generated: lost-to-follow-up status logic with custom structures.

Why: domain phrase activated the wrong workflow archetype.

SUBJID propagation

Expected: propagate existing SUBJID from DC to targets.

Generated: new identifier construction from site/subject components.

Why: model inferred ID creation instead of copy-through behavior.

Behavior examples fixed semantics by showing exact input rows, target variables, edge cases and expected outputs.

INTEGRATION FAILURE MODES

When project context was missing, correct logic still failed the framework

Wrong interface

Missing or wrong imports
No traceability decorator
Wrong function signature
Wrong return object

Effect: test harness could not use the code.

Wrong dependencies

Rebuilt shared helpers
Used generic libraries
Missed local utilities

Example: generic date parsing instead of the approved partial-date parser.

Missing contracts

Ref. file schema absent
Metadata shape unclear
Infrastructure pattern hidden

Effect: plausible code diverged from production ground truth.

Project context fixed mechanics by showing project conventions, pytest assets, type definitions, helper signatures, and metadata contracts.