The Repair Loop in Operation

Self-Healing to the Data, Not the Screen: A Field Report from the Reference Implementation

A COMPANION FIELD REPORT

Melvin Fahnestock

Version 1.0 — August 2, 2026

doi.org/10.5281/zenodo.21761800

expectationdrivendevelopment.com

mel@expectationdriven.com

CONTENTS

What this document is

Expectation-driven development verifies an application at its persisted state: a trusted, fixed, versioned **gold state** supplies both the inputs each test scenario enters and the exact persisted result the application must produce; the inputs travel only through the application’s real screens and real command paths, never through a back door; and what the application actually persisted is compared against gold’s expected state, value by value, failing closed on any disagreement. The state can be anything structured enough to declare as gold; in the implementation reported here it is relational, and the records below speak of rows.

That standing acceptance test is **the gate**: the complete, versioned procedure just described, yielding one verdict for one identified build, execution configuration, and specification version — pass only when every covered obligation holds, or fail with an artifact naming the unmet obligation or the exact disagreement. It is what makes the self-healing built on it different

in kind from the familiar sort. A repair loop can only ever heal to the standard of the verdict that stops it: a loop stopped by interface tests heals until the screens behave, and this one heals until **what you enter, derived from gold, produces a result that matches gold**, verified value by value at the stored state itself.

The main paper specifies the gate — the rules, the two lanes that operate it, and the conditions a pass requires; this report is where its results are. It shows the gate in operation on the reference implementation (§9 of the paper), through four records made across the two days ending at publication, on an instrument that had just completed a dedicated optimization, review, and hardening pass covering the runner, both lane drivers, and the test scenarios themselves:

- **Record I — the Proof lane, clean.** One operator command, one driver launch, the full core chain from level 0: every active scenario ended in a first-attempt PASS or a pinned capability skip, in about nineteen minutes of scenario time, with the resulting evidence markers verified gateable at a single commit.
- **Record II — the Validate lane, at volume.** The session the Proof run's markers exist to gate, run immediately after it on the same commit: roughly three-quarters of a million row-comparisons against gold, row by row, value by value (a closure sum, not a distinct-row count — see Record II), with zero failures, zero retries, and an explicitly reviewed scope whose exclusions are enumerated with exit criteria.
- **Record III — the repair loop, under seeded faults.** A separate run the same day in a commit-free mode, against a working tree into which the operator had planted single-token faults in application code. The chain stopped five times on four scenarios; six faults were found and repaired — one of them by a peer sweep before its scenario ran, and two of them stacked in a single scenario, where the second was invisible until the first was fixed.
- **Record IV — isolating the parity gate.** A second seeded run, on publication day, aimed at the method's central claim: seven faults of a single class — the create path disagreeing with the update path about one field — planted where every interface-level test would pass. Every interface-level test did pass, all seven times; the full-row, full-value comparison against gold supplied all seven gate convictions — no interface, contract, validation, or read-path check convicted any of them. Completeness is machine-checked rather than attested: with the seven repairs applied, the working tree was byte-identical to the committed baseline.

The order matters and is stated so it cannot be misread as stitching: the full-volume session was gated by a separate clean chain and completed *before* either seeded demonstration, and both demonstrations then ran in a commit-free debug mode the harness structurally bars from certifying anything.

These four records are the whole of this report's evidence; earlier runs on superseded versions of the instrument exist and are deliberately not carried, so that every result here is contemporary with the instrument that produced it.

The result, in brief

Seven faults were planted in create-path mappings, each disagreeing with its own correct sibling update path about a single field — among them two transposed contact-email columns, and two transposed payment-control flags, the flags that decide whether a vendor's checks print or are held. Every interface-level test passed, all seven times, and no interface, contract, validation, or read-path check convicted any of them. The full-row comparison against gold supplied all seven gate convictions. With the seven repairs applied the working tree was byte-identical to the committed baseline but for a disposable run marker, so the tree is the injection manifest: exactly seven mutations existed, and none survived. Records I and II are the same machinery with nothing to find; Record III is the repair loop against a different fault class. Author-reported, one narrow fault class, no detection rate. Readers wanting the result first should go to Record IV and Appendix D.

Terminology

The repair loop named throughout is the paper's mechanism (§8.2): the agent-operated diagnose-repair-rerun loop consuming the failure signal the gate was built to emit — what practitioners informally call self-healing, a vernacular the paper touches once and this report otherwise sets aside. The gate itself never repairs anything: it refuses and emits evidence; the loop above it diagnoses, repairs, and reruns (§8.3).

`/proof` and `/validate` name the operator commands that invoke the Proof and Validate lanes of the paper. "Gate condition *n*" refers to the numbered acceptance conditions in §3.3 of the paper; "Rule *n*" to the rules of §3.1. The **expected store** holds the materialized expected-state projections (E_v) — the gold of the opening definition, in the form the comparison consumes; the **working store** is the test store the application writes through its declared doors — the actual side of every comparison. The expected store is never written by a run; the working store is never seeded with expected data (Rule 1, §2.3). The corpus in scope runs from reference code sets of a few dozen codes to entity closures in the low six figures: a **master entity** with its **child records**, a **second master entity**, an **extension profile**, and template, structured-document, valuation, and configuration entities. Per-entity volumes are in Appendix B.

Names, identifiers, and quantities throughout have been redacted or bucketed; what was changed and what was preserved exactly is set out in Appendix E.

The pipeline in one page

`/proof` and `/validate` split one guarantee — *the application faithfully persists what a user or an API caller enters, over the oracle's declared scope* — into the paper's two lanes. The split exists for economy, not principle: in a perfect world a single UI lane would carry every case not intended solely for API use, but full-corpus browser execution runs to hours, days, or beyond, to the point of outright impracticality. So Proof drives a deliberately small slice through the real screens — proving the interface honest and capturing the exact shape of the commands it emits — and Validate pushes the whole corpus through the application's command path constrained to those captured shapes: a case matching nothing the interface demonstrated fails rather than being guessed at. One lane proves the screens; the other proves the machinery underneath, at volume, without ever drifting from what the screens actually send. This page is the minimum needed to read the records; every definition on it defers to the paper, which remains authoritative:

	`/proof` (Proof lane)	`/validate` (Validate lane)
Proves	The declared UI obligations over the versioned coverage model + the captured UI command contract ($K_{b,r,v}$)	Parity over the full in-scope corpus through the official command path, below the declared capture boundary
Volume	Minimum coverage-selected row set exercising every declared UI behavior	Every in-scope row in the expected store
Mechanism	Browser automation drives the real UI one row at a time under deliberately shrunk "proof geometry"	Command-stream replay through the real application-service and domain-validation path
Produces	Fresh PASS markers + the captured command-contract evidence per scenario	Full row- and field-level comparison of actual vs. expected under the parity contract
Skip tolerance	Zero unpinned — an unexercised declared behavior is FAIL; oracle-pinned capability skips are declared, not tolerated	Zero — a skipped row, rejected command, or unsatisfied contract observable is FAIL; whole-scenario capability skips are marker-backed, carrying the Proof lane's pinning

The lanes interlock mechanically, not by convention:

1. `/validate` **refuses to run** without fresh `/proof` PASS markers matching the current build, scenario-manifest hash, store identity, and runtime profile (gate conditions 6–7; runtime identity and stability are conditions 1 and 12, provenance condition 11; store identity is an implementation-specific evidence binding).
2. Every Validate command must **resolve uniquely to, and conform to, a normalized Proof-captured command variant** under the versioned applicability and instantiation rules — replay cannot drift from what the official UI actually emits (gate condition 8).
3. A `/validate` failure that `/proof` never predicted may motivate review of the Proof coverage model — while also potentially exposing a volume-only or below-boundary defect that a correctly bounded Proof slice was never intended to predict. Either way, the gates force the question; nothing about it is aspirational.

Two invariants every event below bottoms out in:

1. **A PASS requires contract-complete treatment of every observable** in the declared comparison closure: exact comparison after canonicalization, declared structural translation, a named semantic invariant, or an explicit exclusion whose mandatory delegated gate also passes (Rule 3). Matching row counts alone are never enough; the value-by-value comparison against gold is the final arbiter of every PASS.
2. **There is no silent force flag.** The only overrides are explicit, individually named, and recorded in the run's own log. Each subsequent pass occurred only after the relevant prerequisite or defect was corrected through its governed path; the one deliberate override in Record II is logged, argued on the record with the reason the refusal's own default suggestion would have been wrong, and re-verified afterward by an independent check. It overrode a preflight coverage check, not marker identity, scenario verdicts, or the final gateability check.

Record I — the Proof chain, clean

Result. One operator command, one driver launch, the full core chain from level 0: 42 active scenarios across a fifteen-level dependency ladder — 36 first-attempt PASS, 6 pinned capability skips, zero failures, zero retries, zero fix commits, roughly nineteen minutes of scenario time. A dedicated check then verified all 42 markers at a single commit. Appendix A carries the chain one row per scenario.

Nothing was retried, and nothing was sampled. Every execution ran at attempts=1; no automatic retry fired anywhere, so no pass is a retried pass. The two long poles — the root reference scenario at about two minutes, the cross-cutting grid sweep at about four — are volume, not instability. Every datum is typed through the real UI: "proof geometry" shrinks the interface to page sizes and viewports of a few rows, so paging, scroll-into-view, and search collisions are all forced to fire — the coverage model's behaviors are forced, not sampled (Rule 7).

The skip tally is the oracle's, not the runner's. Six scenarios reported SKIPPED, five on one dark capability module and one on another; each is pinned in the reviewed expected-skips oracle, resolved against the application's live flag state, and reported in the tally separately from the passes. The oracle's contract makes drift loud in both directions: an unpinned skip — coverage silently shrinking — halts the chain, and so does a pinned scenario unexpectedly running. Five further scenarios are statically deferred, excluded before the chain starts and visible in the discovery record rather than the tally: two whose writes are append-only event streams with no command surface to replay, and three native-transition variants awaiting rebuild on the test-reference pattern, their expected-state seeding removed as an invalid concept — the harness must never write gold; expected state derives from gold. A deferral is not a skip: it never enters the chain. PARTIAL is the contract's word,

not a hedge: the terminal line reads PARTIAL because pinned skips and deferrals sit in range, not because anything went wrong. The tally never folds a skip into a pass.

The markers are the point. That check — a PASS or pinned-SKIP marker (a skip whose required capability is feature-flagged off, in this case) for every one of the 42, minted at the current commit, one commit, no drift — is what Record II stands on, and it is not a formality: markers age by commit, not by clock, so any commit landing after a chain completes strands its markers behind the build they attest, and a session gated on them is refused for commit drift. A clean chain is not durable evidence; a clean chain *at the commit under test* is.

Record II — the Validate session at volume

Result. The session Record I's markers exist to gate: the same invocation's second half, a full-ladder `/validate` on the same commit. 19 replay legs, every one a first-attempt PASS; 6 marker-backed capability skips; 17 further chain scenarios not replayed — 7 classified out under named categories, 10 carrying no distinct replay data — plus the 5 deferred scenarios outside the chain, also classified out. Roughly three-quarters of a million row-comparisons against gold, row by row, value by value; zero failures, zero retries, zero code changes; completed in just over an hour. Appendix B carries the session one row per leg, with per-entity volumes.

The scope is reviewed, and the PASS speaks only for it

Validate and Proof counts are not comparable without the reconciliation, so it comes first, normalized from the driver's own closing line:

> 42 chain = 19 replay + 6 dark-pinned + 7 reported gap + 10 no-distinct-data (silent); 5 deferred classified out but not in the chain.

The PASS certifies the 19 replay members and the 6 dark-pinned skips, and nothing else. Every scenario not replayed carries a named category in a reviewed scope ledger, which also names every gold-populated table not fully validated, each with exit criteria. Condensed from that ledger:

- **Baseline-sync entity.** The root reference entity is written only by its dedicated synchronization path; its application service is query-only, so there is no create command to replay. Nearly every table in the corpus references it, so its durability is exercised indirectly by every downstream replay resolving those references at full volume.
- **Workflow-state machine.** The transaction-document family's gold rows are overwhelmingly terminal states the create command cannot produce — all transaction documents terminal-state, all child documents approved and linked, zero unlinked — and no authorable business key exists at any volume: even the fullest candidate key leaves duplicate groups covering roughly a third of the corpus. By the domain's own rules the front door cannot reproduce gold, so the class is excluded with that reason on the record rather than replayed vacuously.
- **Derived event streams.** The two deferred scenarios whose writes are append-only side effects and framework-seeded rows with no command surface; row replay is not one-to-one with any command.
- **Awaiting rebuild.** The three deferred native-transition variants, pending rebuild on the test-reference pattern.

- **No compare closure.** Behavior proofs that declare no comparison specification; their domain's full-volume compare contract lives on the derived-valuation scenario, which replayed in the six figures.
- **No distinct replay data.** The largest non-replayed class, and why the reconciliation's last term is reported silently. Ten chain scenarios — interface smokes, the always-on dark-capability check, the cross-cutting grid sweep — exercise declared behavior without producing gold rows distinct from those already inside a replayed scenario's closure. They are verified for totality rather than given replay legs that would re-compare rows another leg already compared.

The strongest Validate result on this implementation arrives with its non-coverage enumerated, categorized, and tied to exit criteria. A PASS that will not say what it does not cover is not a verdict.

Two obstacles before the ladder

Both were the machinery working correctly. **One refusal was pre-empted.** A full chain leaves rows from non-replay scenarios restrictively referencing session tables — a derived transaction document's line rows are the canonical instance — and the session's reset preflight refuses to purge what it does not own, so the full ladder cannot satisfy the session's own precondition. So the closure was resolved mechanically — 26 scenarios — and re-proved as a closure-scoped chain before launch. The session runs against closure-scoped proof state, not full-chain state; that is the contract.

One refusal fired. The closure chain did not cover the mainline, because of this invocation's own paperwork: the only commit ahead of the working tree was the Proof run's own run-record commit — documentation only, no source change. The refusal's first suggested remedy, sync forward and re-prove, was the trap: it would have re-minted the 26 closure markers at the new commit while the other 16 active scenarios kept theirs at the old one — a split-commit marker set, precisely the state the gateability check exists to refuse. The explicit override for proving the tree as-is was taken instead, a logged operator act with the docs-only nature of the uncovered commit stated as the argument, and the gateability check was re-run afterward to confirm all 42 markers still sat at one commit. The machine's default advice was wrong; the override was argued on the record; an independent check verified the outcome.

The session itself

The reset passed in seconds. Nineteen replay legs then walked the dependency levels first-attempt clean, from reference sets in the hundreds of rows to four closures each in the six figures; the per-entity volumes are in Appendix B. Nothing wrote gold: the expected store was read-only for the whole session, the replay writes only the working store, and the comparison is one-directional.

Two disclosures on the volume figure. It is the driver's cumulative sum over per-scenario comparison closures, not a count of distinct rows in the working store — a table appearing in more than one scenario's closure is counted more than once. And wall clock concentrated where rows are expensive, not merely numerous: the heaviest leg, the template-entity scenario, took roughly twenty-six minutes for a six-figure closure, about twice the per-row cost of the next slowest leg — recorded as a follow-up observation about replay cost, not a defect.

The evidence tether (conditions 6–8, 11) held end to end: the session gated on markers minted at the commit under test, verified gateable before launch and re-verified after the closure re-proof, and every replayed command resolved to a Proof-captured variant. Nothing was wrong, so the gates had nothing to refuse. That is the steady state, and these two records put it on the record for both lanes, same commit, same day.

Record III — the repair loop, under seeded faults

Result. The two clean records show the machinery with nothing to find; this one shows it finding things. Same day, in the mode built for exactly this: the operator planted six single-token faults in application code, the chain stopped five times on four scenarios, and all six were found and repaired — one by a peer sweep before its scenario ran, and two stacked in a single scenario, where the second was invisible until the first was fixed. Every fix was retested by the driver before the chain advanced; every attempted scenario ended passing; roughly an hour end to end, including all diagnosis and repair. Appendix C records each stop's planted fault, conviction evidence, and repair.

The design

The mode is commit-free by contract. The debug variant of `/proof` authors no commits, pushes nothing, merges nothing; its markers are minted in a scratch worktree, rest on uncommitted code, cannot reach the mainline, and cannot gate a Validate session there. Every repair it makes remains uncommitted, for the operator to inspect or discard. An exercise that deliberately plants defects in the system under test is thereby structurally prevented from contaminating the evidence chain — it cannot produce a green that anything downstream is entitled to trust.

The faults were planted by the operator, into the working tree before launch, as uncommitted single-token mutations in application code. The debug contract treats the working tree as untouchable operator state and bars version control as diagnostic evidence — no diff, no status, no history, no inventory of modified files — so every fault had to be convicted from failure artifacts, then observed behavior, then the source as it stands. Live probes cited below are probes of the running application, not of the repository. The operator's manifest for the exercise: six faults planted; six found; none surviving.

The run

Checkpoint resumes throughout, so five stops cost five checkpoint retests rather than five chains from level 0. Every stop minted a repair debt the moment it fired, tagged to the run's commit-free contract, and every retest cleared it on the record — no debt was outstanding at the end. One evidentiary gap is on the record rather than smoothed over: all four failing scenarios inherited application servers already running from earlier in the chain, so no per-scenario server logs exist for any stop. The interface snapshots and the traces carried every diagnosis — in the stacked fault below, the trace's own request record is what separated "the second submission never happened" from "the second submission was refused."

The spine of the run, as the operator's console relayed it — normalized: roles for names, durations rounded, status lines faithful to the originals:

```
proof  1/42 · L0  root reference scenario          — PASS (~2 min)
proof  2/42 · L1  reference set C                  — PASS (~30 s)
        … fifteen more first-attempt passes, levels 1 through 4 …
proof 18/42 · L4  structured-document scenario     — FAIL (~55 s)
⛔ CHAIN STOPPED — finalStatus=FAIL (only a clean PASS advances)
   diagnosis from artifacts and live probes; one-token repair applied,
   uncommitted; peer sweep repairs the same fault in a second grid service
proof resumed · retesting the failed scenario first
proof  1/25 · L4  structured-document (retest)     — PASS (~75 s) — fix accepted
        … level 5 clean …
proof  6/25 · L6  child transaction document       — FAIL (~40 s)
⛔ CHAIN STOPPED — approval gate inverted; one-token repair, uncommitted
proof  1/20 · L6  child transaction doc (retest)   — PASS (~70 s) — fix accepted
        … level 7: passes and the six pinned dark-module skips …
proof 10/20 · L7  rate-resolution: applicable rate — FAIL (~10 s)
⛔ CHAIN STOPPED — wrong identifier bound; one-token repair, uncommitted
proof  1/11 · L7  rate-resolution (retest)         — PASS (~50 s) — fix accepted
        … siblings on the same endpoint pass; levels 11 and 12 clean …
proof  9/11 · L13 workspace console scenario       — FAIL (~20 s)
⛔ CHAIN STOPPED — effective-date filter off by one day; repair, uncommitted
proof  1/3  · L13 workspace console (retest 1)     — FAIL (~85 s)
   different test, further along — 17 of 35 passing, up from 10;
   same-failure detector: not triggered; progress: yes
⛔ CHAIN STOPPED — assembler persisted the wrong identifier; repair, uncommitted
proof  1/3  · L13 workspace console (retest 2)     — PASS (~2 min) — all 35 tests
proof  2/3  · L13 cross-cutting grid sweep         — PASS (~4 min)
proof  3/3  · L14 master-entity compliance view    — PASS (~30 s)
CHAIN COMPLETE — every attempted scenario passing; markers are scratch state
resting on uncommitted code; nothing merged; the mainline gate is unaffected
```

The five stops. What each fault was, and what convicted it. The evidence chain behind every conviction — the probe truth tables, the live-probe symmetry chains, the alternatives ruled out and why — is carried per stop as diagnosis notes under Appendix C, so the record here stays the story of the run.

- **Stop 1 — the structured-document scenario (L4): a filter that demanded both.** The grid rendered "no records found" for a sentinel name that verifiably existed. Live API probes built the truth table that convicted it: the filter conjoined its two searchable columns where either should suffice. One token — and a peer sweep across every grid service found the same conjunction planted a second time, repaired in the same round, before the chain reached its scenario.
- **Stop 2 — the child transaction document (L6): an approval gate inverted.** The detail page's approval controls never rendered: the live gate offered them *only* on already-approved documents, the exact inverse of the contract the scenario states in both directions. One token.
- **Stop 3 — the rate-resolution scenario (L7): the wrong identifier bound.** A request carrying a deliberately invalid identifier came back with the wrong refusal code — "missing" where "invalid" belonged. Live probes convicted the

controller by symmetry: it bound the master entity's identifier into the field reserved for the second master entity's. One token.

- **Stop 4 — the workspace console (L13): a boundary the decoy was built for.** The console resolved the wrong template: the shared effective-date range filter admitted ranges beginning the day *after* the query date. The scenario's fixture holds a decoy template starting exactly one day later, planted for this boundary; it was wrongly admitted and outranked the baseline on the declared tie-break. A decoy built to sit exactly on the boundary was waiting for the fault that needed it.

Stop 5 — the workspace console again: the stacked fault. The retest after stop 4's fix failed on a *different* test — different position, different message, the passing count advanced from 10 to 17 — and the driver's same-failure detector explicitly did not trigger: progress on the record, not recurrence. The second fault: the console's assembler persisted the outer selector's identifier into the inner selector's field. The stored value was real, typed, and wrong — and invisible to every read. The trace's request record held the conviction: two submissions, ruling out the click never firing — the first succeeding, the second refused outright for matching no template baseline, because the console rebuilds its context from the persisted record and had re-posted the corruption. Two direct submissions of the same context against the live application produced the next two versions correctly, ruling out the versioning path; the corrupted write, re-consumed, was all that remained. The fix uses the correctly resolved local the assembler already had. Rejected alternative: having the test re-enter the context before the second submission — test choreography that would have hidden a persisted-data defect while every consumer that reads the stored value back inherited it. On the final retest all 35 of the scenario's tests passed, and the chain ran out the remaining levels clean.

What the stacking establishes

Stops 4 and 5 are one scenario, two faults, two iterations, and the second fault was undetectable while the first was present, because the console never got far enough to persist and resume. The loop peeled them in order, with the test-coverage count advancing at each iteration as evidence that progress was real rather than asserted. A gate that graded effort would have advanced after the first fix; this one re-ran the scenario, failed it honestly on the newly reachable defect, and advanced only when nothing remained. The write-path fault in the pair is this report's cleanest instance of the paper's central claim: a persisted value that is well-formed, plausible, and wrong survives every check that merely reads it back. What convicted it was the stored state being *re-consumed* by the write path it corrupted — the shape of defect Gold-State Parity™ exists to make loud.

What this record does not establish

Every repaired defect here was planted, and every planted fault was a single-token mutation in application code — a narrow, legible class. This record establishes nothing about multi-line faults, faults distributed across components, subtle logic faults that produce plausible wrong answers, or faults planted in the oracle rather than the application. Six faults are not a sample; no detection rate should be read off this exercise. Against the operator's manifest the result is complete — six planted, six found, none surviving — and within it the evidentiary classes still differ: four faults convicted by a chain stop, one convicted after its neighbor's stop exposed it, and one found by a source sweep rather than by any gate. "Found by the sweep" and "convicted by a gate artifact" are different evidentiary claims, and the sweep-found fault is only the former.

One further thing this record establishes, and it matters for the record that follows: every conviction here came from *above* the persistence comparison — an interface assertion, an API contract, a scenario's own probes. That is the upper layers

policing the fault classes that live in their jurisdiction, on the record and demonstrably competent. The next record plants faults built to live outside every jurisdiction but one.

Record IV — isolating the parity gate

Result. Under the same commit-free debug contract as Record III: seven faults of one class — the create path disagreeing with its sibling update path about one field — planted where every interface-level test would pass. Every interface-level test did pass, all seven times. The full-row comparison against gold supplied all seven gate convictions unaided, with not one comparison false positive. Nine driver segments, eight stops: seven parity convictions and one integrity refusal. With the seven repairs applied, the working tree was byte-identical to the committed baseline except for the disposable run marker. Appendix D records each stop's planted fault, the diff signature that convicted it, and the repair.

Record III supplies the positive control: under the same chain discipline and the same contract, the upper layers convicted faults within their declared jurisdictions. That rebuts the reading that those layers were inert or incapable; it does not make them exhaustive, and this record does not need them to be. It changes the fault class instead — to mutations constructed beyond those layers' assertions but squarely within the full-row comparison's — and asks the question the paper stakes itself on: when every check above the final persistence comparison properly passes, does parity refuse?

One fault class, by design

All seven faults have the same shape: **the create-path mapping disagrees with its sibling update-path mapping about one field.** A discarded input replaced by a hardcoded constant. An active flag forced true. A description fabricated from the neighboring name field. Two contact-email columns transposed. Two payment-control flags transposed. A dimension column filled with the adjacent catalog column's values. A description hardcoded to empty. In every case the update path mapped the same field correctly.

The class is chosen to be invisible everywhere except the stored row. The interface tests assert the submitted command and the resulting row count — both correct in all seven cases, because the command carried the right values and every row was created; the defect lives strictly between the accepted command and the persisted column. Reading the record back shows a plausible value. And because the update path is correct, **editing the record and saving would silently heal it** — the class conceals itself even from the user who goes looking. A defect that only ever exists in rows nobody has edited is the paper's opening example made operational, and the only witness left standing is the persisted row against gold.

Two of the seven make the stakes concrete without leaving role vocabulary: the master entity's two contact-email columns were transposed, and the extension profile's two payment-control flags — the flags that decide whether a vendor's checks print or are held — were transposed. Neither would ever fail a screen. Both are the kind of wrong that surfaces months later, in the mail.

The run

The launch gate fired first: the mainline tree carried uncommitted files, and the run refused to start — a run over a dirty tree would have attributed its repairs to an ambiguous baseline. The operator committed; the clean commit became the baseline under test. Every attempt count in the run is 1. Every stop minted a debt tagged to the commit-free contract and every retest cleared it; the chain finished with every attempted scenario passing.

```
proof  1/42 · L0  root reference scenario     — PASS (~2 min)
        … three more first-attempt passes …
proof  5/42 · L1  reference set D             — FAIL: 7 differences, 7 rows vs 7 rows
⛔ CHAIN STOPPED — interface test PASSED; the compare convicted a hardcoded
   display-order constant on every created row; one-token repair
proof retest · reference set D — PASS — fix accepted, chain advances
⛔ reference set E   — 1 difference: the single inactive reference row came back
   active — a forced-true flag, invisible on the ten rows that default true
⛔ reference set F   — 7 differences: every description exactly its row's name,
   against references that hold none — a fabricated value, not a discard
⛔ master entity     — 12 differences in mirrored pairs: two contact-email
   columns transposed, the nulls travelling with the swap
⛔ extension profile — 6 mirrored boolean differences: two payment-control
   flags transposed; the fourth profile invariant because its flags are equal
⛔ second master entity — a dimension column carrying the adjacent catalog
   column's values; the four sibling comparison specs all PASS
⛔ retest refused    — checkpoint restore hash mismatch: unprovable state;
   the runner's own remedy taken verbatim — plain relaunch from level 0
proof relaunch · fifteen consecutive first-attempt passes re-witness every
   repair below level 4 at the same baseline
⛔ template entity   — 1 difference: the one line carrying a description came
   back empty — a hardcoded null on the line create path
proof retest · template entity — PASS — fix accepted
        … levels 4 through 14 run out clean …
CHAIN COMPLETE — every attempted scenario passing; markers are scratch state;
nothing merged; the mainline gate is unaffected
```

Seven for seven, every gate conviction came from the comparison. Not once did a test, a guard, a validation rule, or any read-path check object. The diff signatures did the diagnostic work on their own: a constant across all rows convicts a literal; a single diff on the one row that differs from the default convicts a forced default; mirrored pairs convict a transposition — with the unaffected row whose two values happen to be equal serving as the control an inversion could not explain; values that duplicate a neighboring column convict a cross-field copy. In every case the update path, mapping the same field correctly, located the fault to the create path before the source was ever opened.

Layer	Verdict on the seven faults
Interface tests (submitted command, row count)	passed all seven
Command-contract conformance	convicted none
Domain validation	convicted none
Read-path rendering	convicted none
Full-row parity against gold	**convicted all seven**

The four upper rows say only what the run supports. The interface tests passed; no check above the comparison convicted anything. “Convicted none” is not a claim that each of those layers ran an assertion capable of catching this class — the class

was built so that none of them would. Seven faults of one shape are not a sample, and no detection rate follows from the table.

What the loop heals to

A repair loop is defined by the signal it heals against. A loop driven by interface tests heals until the screens behave — and on these seven defects it would never have opened at all: the screens behaved throughout, the tests were green, green is that loop's definition of done, and the corrupted values would have shipped inside it. This loop heals against the persisted state itself, so its definition of done is different in kind — not *the interface works* but *what was entered is what is stored*, verified value by value against a truth the implementing agent cannot edit. That is the trust the gate's green actually buys, and the repair loop inherits it whole: a loop can only ever repair to the standard of the verdict that stops it.

The detector that had to learn its own lesson

After the first conviction, a blast-radius sweep looked for other instances — and was wrong twice before it worked; what each failed detector missed, and why, is in the diagnosis note under Appendix D. The detector that finally worked describes the fault *class*: flag any create-path assignment whose right-hand side references no input field, or references an input field other than the one being assigned. Run across every mapping, it returned ten hits: the four remaining planted faults — **predicted before the chain reached any of them** — and six flagged non-faults, each examined and correct by design. The recorded lesson: a regex over one fault's shape is not a blast-radius check; the detector has to describe the class, and the class here is "the create path disagrees with the update path about a field."

The predicted faults were deliberately left in place. From the third conviction onward the loop knew where the remaining four faults were, and touched none of them until the chain convicted each one on its own evidence. That cost four extra stop-repair-retest cycles, and it preserved the only measurement this run exists to take: whether the gates convict each fault unaided. They did — all seven comparison convictions, with not one comparison false positive.

The refusal in the middle, and completeness machine-checked

One stop was not a conviction. A checkpoint resume restored state whose full-scope hash did not match the hash recorded when the checkpoint was taken, and the runner refused to run against state it could not vouch for — eight seconds, nothing exercised, the remedy named in the refusal itself: relaunch plainly from level 0. The remedy was taken verbatim rather than worked around; the rejected alternative — hunting for an older checkpoint that happened to restore cleanly — would have substituted convenience for the verification the gate exists to provide. The relaunch re-proved every level below the failure with fifteen consecutive first-attempt passes, re-witnessing the earlier repairs at the same baseline in the process. The cost of integrity was one segment; the alternative was a tally resting on unprovable state.

The faults were planted as working-tree mutations against a clean committed baseline. When the run ended, the working tree — with all seven repairs applied — was **byte-identical to that baseline** except for the disposable run marker. That identity is a machine-checkable completeness proof: exactly seven mutations existed, all seven were found, and every repair restored the original code exactly. No injection manifest needs to be taken on faith; the tree is the manifest. It also means the committed baseline itself was never defective — the faults existed only as working-tree state for the duration of the run, and nothing this run repaired stands as a defect against any commit.

What this record establishes, and what it does not

On this class — single-line create-path mapping faults whose update paths are correct — the gold-state comparison convicted seven of seven unaided, where every interface-level check passed and where the class self-conceals on edit. The single shape is a design choice, not a limitation to apologize for: it isolates the claim under test, that full-row parity against gold catches what command-level and interface-level assertion cannot. It is also still one class. Seven faults of one shape establish nothing about distributed faults, logic faults with plausible outputs, or faults in the oracle, and no detection rate should be read off a run whose faults were built to be caught by exactly one layer. The run was commit-free throughout; its markers gate nothing.

What would refute it. A conforming reproduction in which every upper layer passes, the persisted state differs from gold inside the declared comparison closure, and parity nonetheless returns green refutes the result directly. A conviction arising on an unmutated baseline would challenge the comparison's own validity: a conviction has to come from a planted fault and not from the comparison's noise, and this run recorded none. A replication should preserve three things: a clean run on the unmutated tree, mutations sealed before the run by someone other than the operator who diagnoses them, and retained artifacts for both directions of adjudication, the false green and the false red. Nothing here is offered as standing outside that test.

Scope and limitations

This is four records from one reference implementation, made across two consecutive days on one hardened build line, reported by the implementation's author. Identifying detail is deliberately generalized as described in Appendix E, and the underlying run artifacts are proprietary and unpublished (see the paper's disclosures) — the report's evidentiary weight is that of a disclosed practitioner account, not an independently auditable dataset.

The clean records contain no defect, and the seeded-fault records contain no organic defect. Records I and II demonstrate the steady state: both lanes green, every gate armed, nothing to refuse. Records III and IV demonstrate the repair loop and the parity gate — against faults the operator planted, thirteen across the two exercises. This report therefore contains no organically discovered defect, and it claims none. What stands behind the assertion that the gates catch what they are aimed at is the seeded evidence of Records III and IV, within the fault-class limits stated in each.

The runner is the instrument, and the instrument's review was not independent. The optimization and hardening pass that preceded these runs was extensive, and it was the author's. The runner has not been independently reviewed, tested against its written specification by anyone other than its author, or validated against an independently authored, sealed known-answer corpus. The seeded exercises are the only runs in which the correct verdicts were known to the operator before the runs began — the only tests of the instrument rather than through it — and they are two exercises, on two deliberately narrow fault classes. The failures in Record III were carried through diagnosis against evidence outside the runner — live application probes, traces, the store itself — while the passes in Records I and II rest on the instrument's word. What would raise confidence is specific and unglamorous: independent review of the comparison and marker paths, specification-based tests written by someone with no stake in the implementation, and seeded exercises with sealed priors and retained artifacts, run by someone other than the operator who planted the faults.

The Validate PASS speaks for its reviewed scope and nothing else. The classified-out categories are enumerated above and in Appendix B, with exit criteria in the scope ledger. The excluded classes are excluded with reasons on the record, not silently.

The volume figure is a closure sum, not a distinct-row count, as disclosed in Record II.

Orchestration and cost figures measure this implementation's tooling on this hardware, not the method's inherent cost. The report is offered as a demonstration that the specified machinery operates as written — not as a controlled study of repair quality, convergence, or economics, which remain the follow-up work of §9.

Closing

One instrument, hardened and frozen; one day of evidence on it. The Proof lane walked its full chain first-attempt clean and minted markers a dedicated check certified at a single commit. The Validate lane, gated on exactly those markers, drove the full in-scope corpus through the application's own front door and stood three-quarters of a million row-comparisons against gold without finding anything to refuse — after pre-empting one launch refusal and taking another head-on, declining to begin until its preconditions were genuinely met, against advice that would have quietly split the evidence. And the same machinery, pointed at a tree the operator had salted with faults, in a mode structurally incapable of contaminating anything, stopped five times, repaired six defects including one stacked behind another, and let nothing advance except on a driver-witnessed retest.

A boring green run in both lanes, on purpose, with the evidence to prove it earned the right to be boring — and, from the same two days, the record of what the same gates do when the code is wrong: first against faults scattered across the application, and then against seven built to be invisible to everything except the comparison itself, which convicted every one.

That is the difference between healing until the screens behave and healing until what you enter, derived from gold, produces a result that matches gold.

Appendix A: the Proof chain, one row per scenario

One verdict line per scenario, as the driver stamped them; roles per the redaction policy, durations rounded. Every row ran at attempts=1. Levels holding no scenarios in this module do not appear.

Level	Scenario (role)	Verdict
L0	root reference scenario	PASS (~2 min)
L1	reference set C	PASS (~30 s)
L1	reference set B	PASS (~15 s)
L1	reference set A	PASS (~15 s)
L1	reference set D	PASS (~20 s)
L1	reference set E	PASS (~20 s)
L1	reference set F	PASS (~20 s)
L2	reference set G	PASS (~20 s)
L2	master entity	PASS (~35 s)
L2	reference set H	PASS (~20 s)
L3	dark-capability scenario (the always-on OFF-side proof)	PASS (~10 s)
L3	extension profile	PASS (~35 s)

Level	Scenario (role)	Verdict
L3	second master entity	PASS (~25 s)
L4	duplicate-rejection smoke (reference set G)	PASS (~10 s)
L4	lookup-cache invalidation scenario	PASS (~15 s)
L4	template-entity scenario	PASS (~30 s)
L4	entity-group scenario	PASS (~30 s)
L4	structured-document scenario	PASS (~25 s)
L5	entity-group clone smoke	PASS (~10 s)
L5	entity-group edit-save smoke	PASS (~10 s)
L5	entity-group readiness-block smoke	PASS (~10 s)
L5	transaction document	PASS (~40 s)
L6	child transaction document	PASS (~30 s)
L7	derived-valuation scenario	PASS (~35 s)
L7	bulk-input resolver scenario	**SKIPPED** — dark module M1, pinned
L7	hold-placement UI smoke	**SKIPPED** — dark module M2, pinned
L7	transaction-document detail-tabs smoke	PASS (~15 s)
L7	negotiation-flow scenario (base)	**SKIPPED** — dark module M1, pinned
L7	negotiation-flow scenario (variant)	**SKIPPED** — dark module M1, pinned
L7	seed-data scenario	**SKIPPED** — dark module M1, pinned
L7	hub scenario	**SKIPPED** — dark module M1, pinned
L7	rate-resolution: applicable rate	PASS (~10 s)
L7	rate-resolution: bridge	PASS (~15 s)
L7	rate-resolution: legacy/future	PASS (~10 s)
L7	child-transaction approval smoke	PASS (~15 s)
L11	configuration-entity scenario	PASS (~30 s)
L12	defaults-profile scenario	PASS (~30 s)
L13	aggregate-worksheet scenario	PASS (~20 s)
L13	derived transaction documents	PASS (~20 s)
L13	workspace console scenario	PASS (~70 s)
L13	cross-cutting grid sweep	PASS (~4 min)
L14	master-entity compliance view	PASS (~30 s)

```
CHAIN COMPLETE — PARTIAL
42 active scenarios: 36 PASS + 6 pinned capability-skips (5 on dark module M1, 1 on M2)
5 statically deferred, excluded at discovery · 0 FAIL · 0 retries consumed
GATEABLE: all 42 active markers PASS/SKIPPED at one commit
```

Statically deferred, visible in the discovery record rather than the tally: an account-provisioning scenario and a message-thread scenario at L3, both append-only event streams with no command surface; and native-transition variants of the child transaction document (L6), the transaction document (L7), and the compliance view (L14), awaiting rebuild on the test-reference pattern, their expected-state seeding having been removed as an invalid concept.

Appendix B: the Validate session, one row per leg

The session's closing evidence box, normalized: roles per the redaction policy, row counts bucketed, durations rounded. Every replay leg passed on its first attempt.

Level	Scenario (role)	Result	Rows compared	Duration
L1	reference set C	PASS	a handful	seconds
L1	reference set B	PASS	several hundred	seconds
L1	reference set A	PASS	~100	seconds
L1	reference set D	PASS	~150	seconds
L1	reference set E	PASS	a handful	seconds
L1	reference set F	PASS	dozens	seconds
L2	reference set G	PASS	~300	seconds
L2	master entity (with child records)	PASS	over ten thousand	~1 min
L2	reference set H	PASS	dozens	seconds
L3	extension profile	PASS	several thousand	~30 s
L3	second master entity	PASS	tens of thousands	~2 min
L4	template-entity scenario	PASS	low six figures	~26 min
L4	entity-group scenario	PASS	tens of thousands	~10 s
L4	structured-document scenario	PASS	low six figures	~4 min
L7	derived-valuation scenario	PASS	low six figures	~17 min
L7	bulk-input resolver scenario	SKIPPED — M1 off, marker-backed	—	seconds
L7	hold-placement UI smoke	SKIPPED — M2 off, marker-backed	—	seconds
L7	negotiation-flow scenario (base)	SKIPPED — M1 off, marker-backed	—	seconds
L7	negotiation-flow scenario (variant)	SKIPPED — M1 off, marker-backed	—	seconds
L7	seed-data scenario	SKIPPED — M1 off, marker-backed	—	seconds
L7	hub scenario	SKIPPED — M1 off, marker-backed	—	seconds
L11	configuration-entity scenario	PASS	low six figures	~9 min
L12	defaults-profile scenario	PASS	thousands	~30 s
L13	aggregate-worksheet scenario	PASS	tens of thousands	~2 min
L14	master-entity compliance view	PASS	tens of thousands	~2 min

```
VALIDATE SESSION COMPLETE (REVIEWED SCOPE) — 19/25 PASS, 6 SKIPPED
scope: 42 chain = 19 replay + 6 dark-pinned + 7 reported gap + 10 no-distinct-data (silent)
plus 5 deferred classified out but not in the chain
roughly three-quarters of a million row-comparisons against gold through the
application command path, row by row, value by value
0 FAIL · 0 retries · 0 code changes · session reset PASS
```

Classified out of replay scope, by category: the root reference scenario (baseline-sync entity); the transaction document, child transaction document, and derived transaction documents (workflow-state machine); the account-provisioning and message-thread scenarios (derived event streams — deferred); the three rate-resolution scenarios (no compare closure); the three native-transition variants (awaiting rebuild — deferred). Categories and exit criteria are held in the reviewed scope ledger described in Record II.

Appendix C: the seeded-fault run, one row per execution

Commit-free mode; all repairs uncommitted by contract; roles per the redaction policy, durations rounded. One row per execution, in execution order across the run's six driver segments; a scenario that failed and was repaired appears again as a retest, so the four failing scenarios occupy more than one row each. The **Segment** column names which driver segment carries the row; each of the five resumes restored the last verified checkpoint rather than re-proving from level 0, so no stop cost a full relaunch. Five scenarios are statically deferred and never entered the chain; they are the same five listed under Appendix A.

Level	Scenario (role)	Verdict	Segment	Fault / fix note
L0	root reference scenario	PASS (~2 min)	1	
L1	reference set C	PASS (~30 s)	1	
L1	reference set B	PASS (~15 s)	1	
L1	reference set A	PASS (~15 s)	1	
L1	reference set D	PASS (~20 s)	1	
L1	reference set E	PASS (~20 s)	1	
L1	reference set F	PASS (~20 s)	1	
L2	reference set G	PASS (~20 s)	1	
L2	master entity	PASS (~35 s)	1	
L2	reference set H	PASS (~20 s)	1	
L3	dark-capability scenario	PASS (~10 s)	1	
L3	extension profile	PASS (~35 s)	1	
L3	second master entity	PASS (~25 s)	1	
L4	duplicate-rejection smoke (reference set G)	PASS (~10 s)	1	
L4	lookup-cache invalidation scenario	PASS (~15 s)	1	
L4	template-entity scenario	PASS (~30 s)	1	

Level	Scenario (role)	Verdict	Segment	Fault / fix note
L4	entity-group scenario	PASS (~25 s)	1	
L4	structured-document scenario	**FAIL** (~55 s)	1	**Stop 1** — grid free-text filter conjoined its two searchable columns. One-token repair, uncommitted; the peer sweep repaired the same fault in the defaults-profile scenario's service at L12 in the same round
L4	structured-document scenario (retest)	PASS (~75 s)	2	fix accepted
L5	entity-group clone smoke	PASS (~15 s)	2	
L5	entity-group edit-save smoke	PASS (~10 s)	2	
L5	entity-group readiness-block smoke	PASS (~10 s)	2	
L5	transaction document	PASS (~40 s)	2	
L6	child transaction document	**FAIL** (~40 s)	2	**Stop 2** — approval controls gated on the approved status where open belonged. One-token repair, uncommitted
L6	child transaction document (retest)	PASS (~70 s)	3	fix accepted
L7	derived-valuation scenario	PASS (~35 s)	3	
L7	bulk-input resolver scenario	**SKIPPED** (seconds) — dark module M1, pinned	3	
L7	hold-placement UI smoke	**SKIPPED** (seconds) — dark module M2, pinned	3	
L7	transaction-document detail-tabs smoke	PASS (~15 s)	3	
L7	negotiation-flow scenario (base)	**SKIPPED** (seconds) — dark module M1, pinned	3	
L7	negotiation-flow scenario (variant)	**SKIPPED** (seconds) — dark module M1, pinned	3	
L7	seed-data scenario	**SKIPPED** (seconds) — dark module M1, pinned	3	
L7	hub scenario	**SKIPPED** (seconds) — dark module M1, pinned	3	
L7	rate-resolution: applicable rate	**FAIL** (~10 s)	3	**Stop 3** — the master entity's identifier bound into the field reserved for the second master entity's. One-token repair, uncommitted
L7	rate-resolution: applicable rate (retest)	PASS (~50 s)	4	fix accepted
L7	rate-resolution: bridge	PASS (~15 s)	4	sibling on the same endpoint; covers the stop 3 fix
L7	rate-resolution: legacy/future	PASS (~10 s)	4	sibling on the same endpoint; covers the stop 3 fix

Level	Scenario (role)	Verdict	Segment	Fault / fix note
L7	child-transaction approval smoke	PASS (~15 s)	4	
L11	configuration-entity scenario	PASS (~35 s)	4	
L12	defaults-profile scenario	PASS (~30 s)	4	first-attempt PASS carrying the sweep-found repair, applied before the chain reached it
L13	aggregate-worksheet scenario	PASS (~20 s)	4	
L13	derived transaction documents	PASS (~20 s)	4	
L13	workspace console scenario	**FAIL** (~20 s)	4	**Stop 4** — shared effective-date range filter admitted ranges starting the day after the query date. Repair, uncommitted
L13	workspace console scenario (retest 1)	**FAIL** (~85 s)	5	**Stop 5**, the stacked fault — the assembler persisted the outer selector's identifier into the inner selector's field. 17 of 35 tests passing, up from 10; same-failure detector not triggered. Repair, uncommitted
L13	workspace console scenario (retest 2)	PASS (~2 min)	6	fix accepted; all 35 tests passing
L13	cross-cutting grid sweep	PASS (~4 min)	6	
L14	master-entity compliance view	PASS (~30 s)	6	

The stops in detail. The same five stops with the evidence that convicted each fault and the repair that cleared it.

Stop	Level / scenario	Planted fault	Convicted by	Repair, retested
1	L4 structured-document	Grid free-text filter conjoined its two searchable columns (AND for OR)	Live-API truth table: both-columns terms matched, single-column terms returned zero rows	One token; retest PASS
—	L12 defaults-profile	Same conjunction, second grid service	A peer sweep across every grid service, before the scenario ran	Same round; first-attempt PASS at L12
2	L6 child transaction document	Approval controls gated on the approved status where open belonged	The scenario's own probes: all documents proven open; its negative assertion requires the control gone once approved — the gate was the exact inverse	One token; retest PASS
3	L7 rate-resolution (applicable rate)	Master entity's identifier bound where the second master entity's belonged	Live-probe symmetry: the bogus value produced the invalid-entity refusal in one parameter and "missing" in the other	One token; retest PASS; sibling scenarios covered the fix
4	L13 workspace console (iteration 1)	Effective-date range filter admitted ranges starting the day after the query date	The fixture's decoy template — planted to start exactly one day after — wrongly admitted and outranking the baseline on the tie-break	One token; retest advanced 10 → 17 tests, then failed on the stacked fault
5	L13 workspace console (iteration 2)	Assembler persisted the outer selector's identifier into the inner selector's field	The console resuming from its own persisted state: the re-posted assemble refused for matching no baseline — the corrupted write re-consumed by the write path	One token; final retest all 35 tests PASS; chain ran out clean

Diagnosis notes. The evidence chain behind each conviction, held here so Record III can carry the run and this appendix can carry the forensics. Stop 5's diagnosis stays in the record: it is the stacked pair, and the peeling is the point.

Stop 1 — the structured-document scenario (L4): a filter that demanded both. The grid rendered "no records found" for a sentinel name that verifiably existed. Live API probes built the truth table: the shared sentinel prefix, in both name and

description, returned all six rows; the full name, in the name only, returned zero; a description-only term returned zero, ruling out "searches the wrong column"; every both-columns substring passed and every single-column substring failed, down to a two-character term matching exactly the one row carrying it in both. The filter conjoined its two searchable columns where either should suffice. One token. A peer sweep then examined every grid service's filter predicate and every conjunction within them: ten services carried one in their text, eight were null-guards inside a disjunct and correct, and **exactly two were genuine — this one and the defaults-profile scenario's service at L12** — fixed in the same round; that scenario later passed on its first attempt, its planted fault repaired before the chain ever reached it. Rejected alternative, on the record: rewriting the test's row locator. It is the suite's canonical row selector, and the same test had just driven pagination and multi-sort through it successfully — innocent; only the narrowed search made it resolve to nothing.

Stop 2 — the child transaction document (L6): an approval gate inverted. The detail page's approval controls never rendered. The test's own preceding probe had just proven every created document open, and its later assertion requires the approval control to vanish once a document is approved; the live gate offered the controls *only* on already-approved documents — the exact inverse of the contract the scenario states. The same block corroborates: it also renders the submit-for-review control, coherent on an open record and incoherent on an approved one, and the page's own attachment guard already treats approved and denied as terminal. The fix moves the gate from the approved status to the open status, satisfying both directions of the stated contract. Rejected alternative: pointing the test at the review lane's approval control — a different lifecycle transition on a different endpoint, whose substitution would have deleted the open-to-approved coverage rather than restored it.

Stop 3 — the rate-resolution scenario (L7): the wrong identifier bound. A request carrying a deliberately invalid identifier for the second master entity came back with the wrong refusal code — "missing" where "invalid" belonged. Live probes convicted the endpoint's controller by symmetry: the bogus value in one parameter produced the invalid-entity refusal quoting it, in the other it produced "missing," and a third probe put a bogus value in an unrelated scope parameter and got the correct scope refusal back — not a blanket binding failure but one line, with the resolver's own guards and queries reading innocent on inspection. The controller bound the master entity's identifier into the field reserved for the second master entity's and never read the real parameter. One token. Rejected alternative: accepting the observed status as the contract. The two refusal codes are distinct diagnostics the endpoint exists to separate; accepting either would have made the scenario's whole status contract unassertable. The sibling scenarios on the same endpoint passed on the next launch, covering the fix.

Stop 4 — the workspace console (L13): a boundary the decoy was built for. The console resolved the wrong template. The shared effective-date range filter admitted ranges beginning the day *after* the query date — off by one day — and the scenario's fixture holds a decoy template starting exactly one day after the effective date for precisely this reason; the decoy was wrongly admitted and outranked the baseline on the declared tie-break. The fixture's second decoy — a range ending on the effective date — contained neither the true date nor the shifted one and was correctly excluded either way: exactly why only one decoy leaked, and the discriminating evidence that the shift was one day forward. Every caller of the filter was checked for a compensating shift before the fix; none compensates. Rejected alternative: moving the decoy's window so the shifted date falls outside it, which would have deleted the boundary assertion the decoy exists to make and left a day-shifted resolver live behind every effective-dated query in the console. A decoy built to sit exactly on the boundary was waiting for the fault that needed it.

Six faults, five stops, one sweep-found repair, two faults stacked in one scenario and peeled in order. The chain advanced only on driver-witnessed retests; the run minted nothing that could reach the mainline or gate a session.

Appendix D: the parity run, one row per execution

Commit-free mode; all repairs uncommitted by contract; roles per the redaction policy, durations rounded. Every conviction came from the final comparison; the interface test passed in all seven cases. Diff counts are per-slice structural counts. One row per execution, in execution order across the run's nine driver segments. **Every row ran at attempts=1**, and the integrity refusal at segment 7 ran at attempts=0 — nothing was exercised under it. Segment 8 is a plain relaunch from level 0 rather than a resume, which is why levels 0 through 3 appear twice; those fifteen consecutive first-attempt passes re-witness every repair below level 4 at the same baseline. Five scenarios are statically deferred and never entered the chain; they are the same five listed under Appendix A.

Level	Scenario (role)	Verdict	Segment	Fault / fix note
L0	root reference scenario	PASS (~2 min)	1	
L1	reference set C	PASS (~30 s)	1	
L1	reference set B	PASS (~15 s)	1	
L1	reference set A	PASS (~15 s)	1	
L1	reference set D	**FAIL** (~20 s)	1	**Stop 1** — 7 differences, 7 rows vs 7 rows: a display-order constant on every created row. Interface test PASSED. One token
L1	reference set D (retest)	PASS (~60 s)	2	fix accepted
L1	reference set E	**FAIL** (~20 s)	2	**Stop 2** — 1 difference: the single inactive reference row came back active, a forced-true flag invisible on the rows that default true. Interface test PASSED. One token
L1	reference set E (retest)	PASS (~60 s)	3	fix accepted
L1	reference set F	**FAIL** (~20 s)	3	**Stop 3** — 7 differences: every description exactly its own row's name, against references that hold none. Interface test PASSED. One token
L1	reference set F (retest)	PASS (~55 s)	4	fix accepted
L2	reference set G	PASS (~20 s)	4	
L2	master entity	**FAIL** (~35 s)	4	**Stop 4** — 12 differences in mirrored pairs across six records: two contact-email columns transposed, the nulls travelling with the swap. The two sibling specs PASS. Interface test PASSED. Two lines
L2	master entity (retest)	PASS (~70 s)	5	fix accepted
L2	reference set H	PASS (~20 s)	5	
L3	dark-capability scenario	PASS (~10 s)	5	
L3	extension profile	**FAIL** (~35 s)	5	**Stop 5** — 6 mirrored boolean differences across three of the four profiles: two payment-control flags transposed; the profile whose flags are equal is invariant. Interface test PASSED. Two lines
L3	extension profile (retest)	PASS (~80 s)	6	fix accepted
L3	second master entity	**FAIL** (~25 s)	6	**Stop 6** — 6 differences on one column: a dimension column carrying the adjacent catalog column's values. The four sibling comparison specs all PASS. Interface test PASSED. One token
L3	second master entity	**REFUSED** (~8 s)	7	*Not a fault* — the integrity refusal: restored state did not match the checkpoint's recorded full-scope hash. attempts=0, nothing exercised; the runner's own remedy taken verbatim, a plain relaunch from level 0
L0	root reference scenario	PASS (~2 min)	8	
L1	reference set C	PASS (~20 s)	8	

Level	Scenario (role)	Verdict	Segment	Fault / fix note
L1	reference set B	PASS (~15 s)	8	
L1	reference set A	PASS (~15 s)	8	
L1	reference set D	PASS (~20 s)	8	re-witnesses the stop 1 repair at the same baseline
L1	reference set E	PASS (~20 s)	8	re-witnesses the stop 2 repair
L1	reference set F	PASS (~20 s)	8	re-witnesses the stop 3 repair
L2	reference set G	PASS (~20 s)	8	
L2	master entity	PASS (~35 s)	8	re-witnesses the stop 4 repair
L2	reference set H	PASS (~20 s)	8	
L3	dark-capability scenario	PASS (~10 s)	8	
L3	extension profile	PASS (~35 s)	8	re-witnesses the stop 5 repair
L3	second master entity	PASS (~25 s)	8	the stop 6 fix accepted here, by the relaunch rather than by a resume
L4	duplicate-rejection smoke (reference set G)	PASS (~10 s)	8	
L4	lookup-cache invalidation scenario	PASS (~15 s)	8	
L4	template-entity scenario	**FAIL** (~30 s)	8	**Stop 7** — 1 difference on the line comparison spec: the only reference line carrying a description came back empty. Interface test PASSED. One token
L4	template-entity scenario (retest)	PASS (~65 s)	9	fix accepted
L4	entity-group scenario	PASS (~30 s)	9	
L4	structured-document scenario	PASS (~30 s)	9	
L5	entity-group clone smoke	PASS (~10 s)	9	
L5	entity-group edit-save smoke	PASS (~10 s)	9	
L5	entity-group readiness-block smoke	PASS (~10 s)	9	
L5	transaction document	PASS (~40 s)	9	
L6	child transaction document	PASS (~35 s)	9	
L7	derived-valuation scenario	PASS (~40 s)	9	
L7	bulk-input resolver scenario	**SKIPPED** (seconds) — dark module M1, pinned	9	
L7	hold-placement UI smoke	**SKIPPED** (seconds) — dark module M2, pinned	9	
L7	transaction-document detail-tabs smoke	PASS (~15 s)	9	
L7	negotiation-flow scenario (base)		9	

Level	Scenario (role)	Verdict	Segment	Fault / fix note
		SKIPPED (seconds) — dark module M1, pinned		
L7	negotiation-flow scenario (variant)	**SKIPPED** (seconds) — dark module M1, pinned	9	
L7	seed-data scenario	**SKIPPED** (seconds) — dark module M1, pinned	9	
L7	hub scenario	**SKIPPED** (seconds) — dark module M1, pinned	9	
L7	rate-resolution: applicable rate	PASS (~10 s)	9	
L7	rate-resolution: bridge	PASS (~15 s)	9	
L7	rate-resolution: legacy/ future	PASS (~10 s)	9	
L7	child-transaction approval smoke	PASS (~10 s)	9	
L11	configuration-entity scenario	PASS (~30 s)	9	
L12	defaults-profile scenario	PASS (~30 s)	9	
L13	aggregate-worksheet scenario	PASS (~20 s)	9	
L13	derived transaction documents	PASS (~20 s)	9	
L13	workspace console scenario	PASS (~70 s)	9	
L13	cross-cutting grid sweep	PASS (~4 min)	9	
L14	master-entity compliance view	PASS (~25 s)	9	

The stops in detail. The same seven fault stops and the integrity refusal, with the diff signature that convicted each fault and the repair that cleared it.

Stop	Level / scenario	Planted fault (create path only; update path correct)	Diff signature that convicted it	Repair
1	L1 reference set D	Display-order field hardcoded to a constant	The same value on all seven rows against three distinct reference values	One token; retest PASS
2	L1 reference set E	Active flag forced true, discarding the input	Exactly one diff — the single inactive reference row — always in the forced direction	One token; retest PASS
3	L1 reference set F	Description fabricated from the name field	Every description exactly its own row's name, against references holding none	One token; retest PASS
4	L2 master entity	Two contact-email columns transposed	Twelve diffs in mirrored pairs across six records, nulls travelling with the swap	Two lines; retest PASS
5		Two payment-control flags transposed		Two lines; retest PASS

Stop	Level / scenario	Planted fault (create path only; update path correct)	Diff signature that convicted it	Repair
	L3 extension profile		Six mirrored boolean diffs across three profiles; the equal-flags profile invariant	
6	L3 second master entity	Dimension column filled with the adjacent catalog column's values	All six diffs on one column, each value duplicating its neighbor; four sibling specs PASS	One token; verified by the level-0 relaunch
—	L3 (retest)	*Not a fault:* checkpoint-restore hash mismatch	The runner refused unprovable state, eight seconds, nothing exercised	Plain relaunch from level 0, as the refusal instructed
7	L4 template entity	Line description hardcoded to empty	Exactly one diff — the only reference line carrying a description	One token; retest PASS; chain ran out clean

Diagnosis note — the blast-radius sweep. The sweep that followed the first conviction was wrong twice before it worked. A scan for literal right-hand sides missed the forced boolean; widened to assignments referencing no input at all, it still missed the fabricated description, whose right-hand side references an input — the wrong one. Only a detector written against the fault *class* returned the remaining four, as recorded in Record IV. The per-stop diff signatures that convicted each fault are in the table above; they were sufficient on their own, and no probe chain was needed for any of the seven.

Seven faults, seven convictions by the comparison alone, one integrity refusal honored verbatim. After the repairs the working tree was byte-identical to the committed baseline: the tree is the injection manifest.

Appendix E: the redaction policy

The reference implementation is proprietary and its operator is not identified (see the paper's disclosures). Business-entity labels, scenario names, schema concepts, and tool and product names have been replaced with relational roles; commit identifiers, repository references, file paths, and infrastructure details have been removed; data-record counts, tallies, and durations have been rounded or bucketed so that no combination of quantities can reconstruct or recognize a particular database or repository; small structural process counts that carry method content (scenario and level counts, iteration caps, attempt numbers) are retained deliberately. What is preserved exactly is the causal structure: the sequence of refusals, diagnoses, repairs, stops, and passes, and the verdict of every event. Runner output appears as **normalized excerpts**, structure and verdict lines faithful to the originals, identifiers and quantities neutralized as above.