

Zero-divergence refactors

Legacy replacement does not require a cut-down successor and a decade of feature-parity work. If behaviour can be observed, specified and challenged, a modern system can be built behind an equivalence boundary and moved into service without forgetting its users.

AUTHORED BY

Daniel Johnson & Gary McLean Hall

AI DRAFTING

Astra, Fable & the models of the moment

EVIDENCE CHECKED

15 September 2026. The date tells you when the cited material was reviewed, not when the future agreed to behave.

INVALIDATED BY

When transition evidence or system constraints contradict the proposed equivalence boundary.

EXPIRY DATE

The reader's call. Before "we have a rollback plan" substitutes for rehearsing it.

The founders own the argument; AI helped draft the prose. The sources and limits are here so you can challenge both.

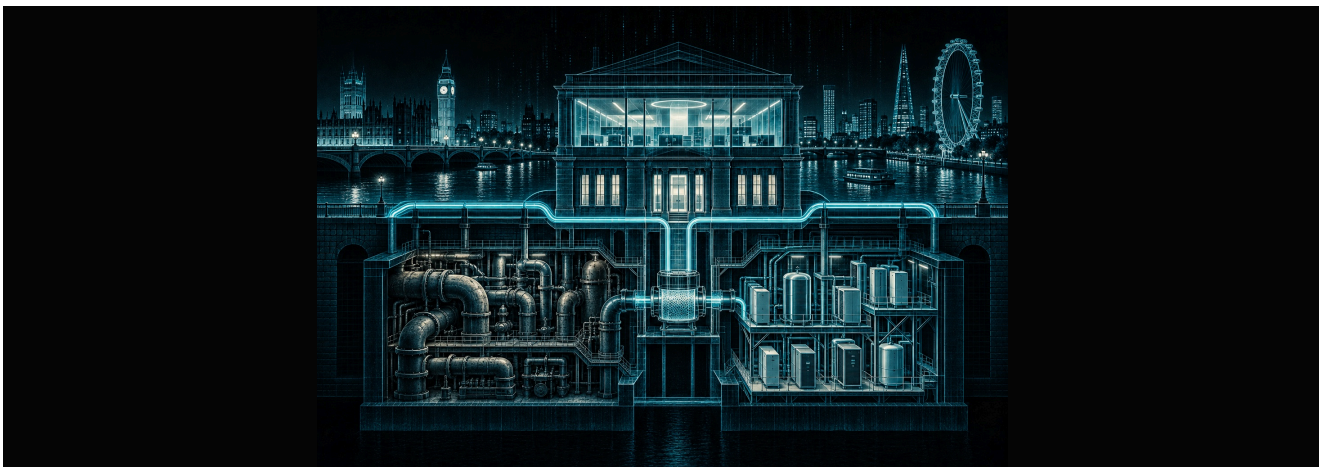
Contents

- 01. The false choice
 - 02. Legacy is accumulated truth and accumulated accident
 - 03. Observe before redesigning
 - 04. Recover domain boundaries
 - 05. Build the whole agreed product
 - 06. Differential evidence
 - 07. Migration is a product capability
 - 08. The release boundary
 - 09. A bounded case: 2,510 observations
 - 10. Public-sector relevance
 - 11. Failure modes
 - 12. What buyers should require
 - 13. Conclusion
 - 14. References
-

This paper combines a proposed replacement method, external research and one bounded Zero Drift evidence receipt. Its review date records editorial and source checks, not a certification of universal equivalence or a measured success rate. Constructed transition examples are distinguished from the recorded census result.

The false choice

Legacy programmes are often framed as a choice between two bad options. Keep the old system and accept rising cost and fragility, or replace it with a smaller “minimum viable” product and spend years rebuilding the capability users already had.



There is a third route. Treat the existing system as evidence. Recover its domains, contracts and behaviour. Build a modern implementation behind a measured boundary. Compare old and new against the same observations. Keep rollback until the evidence supports transition.

Zero divergence is the target within a stated census, not a claim of mathematical equivalence across every possible state. The bounded wording matters. It replaces “we think the migration is safe” with “these observed behaviours, properties and states did not change under these conditions.”

The third route does not promise a costless transition. It moves effort into observation, classification, coexistence and proof. Those activities are valuable precisely because a replacement can otherwise look complete while losing a rare workflow, historical rule or operational capability. The comparison boundary makes that loss detectable before users have to discover it through failed work.

The target also needs a definition of permissible difference. A modern interface may change presentation while preserving the same business outcome. A corrected policy interpretation may intentionally change an outcome. Those are not interchangeable forms

of parity. Record which dimensions must remain equivalent, which may improve and who can approve a changed obligation. Without that distinction, teams either preserve every accident or use “modernisation” as a general excuse for divergence.

Legacy is accumulated truth and accumulated accident

An old system contains years of decisions. Some encode essential domain knowledge. Some are workarounds for retired constraints. Some are defects that users have learned to navigate. Documentation rarely separates them cleanly.

Legacy behaviour needs a disposition

Preserve

An obligation or depended-on behaviour accepted by its owner.

Do not copy blindly

A defect or unknown requires an explicit correction or decision.

Starting from a blank product backlog loses that information. Copying the architecture preserves too much of it. The work is classification.

Sources include code, schemas, configuration, logs, interfaces, reports, support cases, user practice and production observations. Each behaviour is marked as obligation, tolerated legacy, defect to correct or unknown requiring a decision. That classification becomes the replacement specification.

The organisation, not the implementation team, must own choices that change user-visible or legally relevant behaviour.

Classification benefits from concrete examples rather than abstract agreement that the old system is messy. An unusual ordering rule may exist because an external recipient depends on it. A duplicated field may be an abandoned workaround or the only surviving record of a historical decision. Investigate the consequence of removing it and the evidence for preserving it. Code explains what happens; user practice and domain authority help explain which part is an obligation.

Unknown behaviour should remain visibly unknown until resolved. If it affects a critical path, it can block that transition while independent domains continue. Treating unknowns as defects to remove risks deleting hidden requirements. Treating them all as permanent obligations recreates the old system's accidents. The replacement specification should preserve the classification decision and its source so a later reviewer can understand why a strange behaviour survived or was deliberately changed.

Observe before redesigning

Create a behavioural census before changing the path.

A census needs initial state

Recorded request

Useful only alongside relevant relationships, configuration and rule versions.

Sampling question

Which rare, consequential state transitions are missing?

Record representative operations and their durable effects. Include ordinary, boundary, historical and failure cases. Normalise genuinely irrelevant variation such as generated identifiers or timestamps, but document every normalisation. Add explicit properties where examples cannot cover the space.

The census is strongest when it observes through stable external seams: APIs, commands, events and user workflows. Internal implementation details can help diagnose a difference, but coupling the comparison to them makes architectural change harder.

For sensitive systems, observation needs lawful data handling. Production records may need anonymisation, synthetic reconstruction or execution inside controlled environments. Evidence collection is part of the programme, not a reason to copy live data casually.

A useful census describes its composition, not only its size. Countable observations should be grouped by operation, role, state, historical rule and failure mechanism. Thousands of repeated reads cannot substitute for evidence about the one transition that changes an entitlement or sends money. Known gaps should be explicit, particularly where production traffic is rare but the consequence of an error is high.

Replaying observations may require reconstructing their initial state. A request that succeeded against last year's configuration can mean something different against today's database. Preserve the relevant rule versions and relationships, or build a controlled synthetic fixture that represents them. Reproduction does not require retaining unnecessary personal data, but it does require retaining the semantics that made the case informative. Otherwise the census becomes a list of inputs detached from the conditions under which the old system produced its answers.

Recover domain boundaries

Monoliths are difficult to replace because business capabilities, data ownership and deployment are entangled. A modern target should separate domains around reasons to change.

Test a domain boundary

Policy change

Identify the one owner of the rule and its exposed contracts.

Entanglement signal

Unrelated modules edit shared state or must release together.

Identify which capability owns each state and rule. Define contracts at the seams. Avoid a distributed monolith in which services are separate processes but every change still requires coordinated releases and shared tables.

AWS's modernisation guidance describes patterns such as strangler fig and branch by abstraction. Their relevant property is coexistence behind a seam that supports changing the active implementation. Whether reversal remains safe depends on the programme's data and effect contracts. [Strangler fig](#); [branch by abstraction](#).

Domain separation also makes scale selective. The expensive or heavily used capability can scale without multiplying the whole estate. Clear ownership can keep policy changes local when the surrounding contracts do not change.

Domain recovery does not require immediate distribution into separate services. A modular implementation can establish explicit ownership and contracts inside one deployment, then separate operations where evidence supports the additional complexity. The important question is whether a domain can change without uncontrolled consequences, not how many processes appear on an architecture diagram.

Test that claim with a representative policy change. Identify which module owns the rule, which contracts expose its result and which consumers need to react. If unrelated areas must edit shared state or coordinate releases for every change, the boundary is still entangled. This exercise gives the replacement architecture a falsifiable purpose. It should reduce a known constraint of the old system while preserving its agreed behaviour, rather than merely replace one fashionable topology with another.

Build the whole agreed product

A replacement should not quietly redefine "complete" as "what was easy to rebuild".

Parity includes the awkward work

Visible capability

The workflows and reports people already rely on.

Transition capability

Identity, integration, data migration, recovery and operating handover.

Compile the required behaviour into units: identity, data, workflows, integrations, reports, migration, operations and handover. Include awkward cases users depend on. Implement behind stable contracts. Run the new path continuously against the census rather than waiting for a final test phase.

AI changes the economics of this work. Agents can map code, draft adapters, generate fixtures, implement bounded units and investigate divergences. They may make a broader parity programme economical, provided review and repair do not consume the implementation savings. They do not decide which legacy behaviour should survive.

The hypothesis is that cheaper mechanical work can make a complete replacement more feasible. That requires a measured saving across preparation, construction, comparison and repair, rather than an assumption based on code-generation speed.

Differential evidence

Differential testing supplies identical inputs to old and new implementations and compares outputs. For a real system the comparison should include:

Two successes can still diverge

Both responses

The case-closing endpoint returns success.

Different effects

The replacement sends two notifications after a retry; the old path sends one.

- response value and error semantics;
- durable data changes;
- emitted events and notifications;
- permissions and information exposure;
- ordering, idempotency and retry behaviour;
- performance bounds; and
- operator-visible logs and metrics.

Recent research on differential fuzzing for LLM-assisted refactoring explores this same problem: functional equivalence is better assessed through generated executions and observed differences than through code similarity. The field is young, and research

benchmarks do not remove the need for domain-specific oracles.

Dristi and Dwyer's study evaluated six models on three datasets and two refactoring types. It reports functionally non-equivalent outputs and cases missed by the existing suites. That supplies evidence against treating an unchanged test result as sufficient equivalence evidence. It does not estimate the failure probability of a complete enterprise replacement, whose state, integrations and operating conditions extend beyond those tasks. [Dristi and Dwyer, 2026](#).

Every divergence becomes a decision. Fix the new implementation, expand the specification, or approve an intentional change. The decision and evidence remain attached to the unit.

Consider a constructed case-closing workflow. Both implementations return success, but the replacement emits two notifications after a retry while the old path emits one. A response-only comparator reports parity and misses the user-visible difference. The observation should include the durable case state, event identity, recipient and retry semantics. A normaliser may ignore incidental delivery timestamps while preserving the number and relationship of the effects.

When a difference appears, preserve the smallest reproducible case and classify it before changing either oracle. The new implementation may be wrong, the legacy behaviour may be defective, or the comparison may be treating irrelevant variation as meaningful. Each diagnosis has a different remedy. Automatically updating expected output to match the replacement makes the programme easier to pass by removing the very independence that differential evidence was meant to provide.

Migration is a product capability

Data migration is not a one-off script at the end. It is a repeatable capability with reconciliation and rollback.



Profile the source, define ownership and transformation rules, rehearse on representative volume, measure duration and verify invariants. Preserve identifiers and provenance where required. Design for dual read, dual write or event replay only where the risk justifies the complexity.

A migration can be operationally reversible even when every data transformation is not. The rollback plan must say what happens to writes made after transition, how users are routed, and which point makes reversal unsafe.

The cutover criterion is evidence, not calendar pressure.

An illustrative migration rehearsal should include more than a successful bulk import. Start with representative relationships and difficult records, transform them, reconcile counts and invariants, then exercise the application against the migrated state. A table-level count can agree while ownership links or historical versions are wrong. Include rejected records and define whether they stop cutover, enter a repair queue or require a documented exception.

Then introduce writes after transition and test the rollback story. Returning traffic to the old implementation may be unsafe if it cannot represent those writes. The plan may require replay, a reverse transformation, a temporary write restriction or a forward recovery strategy. The appropriate choice depends on the domain and cannot be inferred from the existence of a feature flag. State the irreversible boundary before release, while there is still time to change the transition design.

The release boundary

Strangler-style transition allows capability-by-capability movement. Branch by abstraction can replace an internal implementation while callers remain stable. Both depend on explicit seams and telemetry.

Shadow does not mean harmless

Authoritative path

The old system owns real writes and notifications.

Shadow path

Represent effects for comparison; suppress duplicate live actions.

Shadow execution is useful when the new path can process real-shaped traffic without causing effects. Compare results while the old path remains authoritative. Canary release then exposes a controlled share of genuine use. Feature controls permit rapid return where appropriate.

Do not leave the safety net indefinitely. Parallel systems create cost and inconsistent sources of truth. Define the evidence required to retire the old path and a deadline for unresolved exceptions.

During shadow execution, suppress effects that would duplicate real work. Reading the same input through two paths is not safe if both send messages, update balances or trigger downstream processes. Define which path is authoritative and how the shadow environment represents effects for comparison. Shadow agreement supports a specific claim about the observed path; it does not by itself prove that production writes, operational permissions and recovery work correctly.

A canary adds real use and therefore needs an intervention rule. Identify the signals that pause expansion, who may return traffic and how the team reconciles state afterwards. Retirement is a separate acceptance decision: the old path can be removed when the agreed census, operational evidence and outstanding exceptions support it. Keeping both systems indefinitely is not a neutral default; it leaves continuing cost and the possibility of competing truths.

A bounded case: 2,510 observations

The [26 August 2026 evidence receipt](#) records 2,510 identical observation points across a 193-row Phoenix Ember replacement census, with no recorded divergences, crashes or unexplained one-sided results.

Read the bounded receipt

The published result is agreement within a particular census. The summary receipt does not establish universal equivalence.

Corpus	193 census rows.
Observed	2,510 observation points; 2,510 recorded identical.
Exceptions	0 recorded divergent, crashed or unexplained one-sided.

Recorded 26 August 2026 · summary receipt, not the executable reproduction package.
[Inspect the receipt →](#)

The result establishes agreement only within that census. To judge its release significance, inspect the represented operations and roles, compared state and effects, normalisation rules and evidence that the comparator rejects a seeded material fault. It does not establish universal equivalence, correctness of preserved behaviour or a general success rate for Zero Drift replacements.

Read the receipt as a document, not a score

The published receipt exposes the following fields. These are the recorded figures, not an extrapolation or a visual estimate.

Receipt field	Recorded value	What it establishes
Recorded date	26 August 2026	Date attached to this evidence record
Corpus rows	193	Number of rows represented by the final census
Observation points	2,510	Compared observations within those rows
Identical	2,510	Agreement recorded at those observation points
Divergent	0	No divergence recorded by that comparison
Crashed	0	No crash recorded in the reported run
Unexplained one-sided	0	No unexplained observation present on only one side

The receipt also identifies two source records by name, line count and SHA-256 hash. A hash can help establish that a subsequently supplied file matches the referenced bytes. It cannot demonstrate that the corpus is representative, the normaliser is sound or the preserved result is legitimate. Those are separate questions requiring the underlying method and artefacts. The public JSON is a summary receipt, not the complete executable reproduction package. [Published evidence receipt](#).

Rows and observations are not interchangeable units. A row can contain several observations of the same operation, and the receipt does not claim statistical independence among them. It would therefore be misleading to turn 2,510 agreements into a probability that an arbitrary unseen operation will agree. The useful next question is which obligations those observations cover and which important ones remain outside the census.

Nor does zero divergence establish improvement. Two implementations can agree on a historical defect. The replacement programme still needs a classification of preserved behaviour and approved corrections, with separate evidence for new security, performance or operational claims. The receipt supports a narrow, valuable statement about the comparison performed. Its credibility depends on keeping that statement narrow enough to challenge.

That is the standard replacement programmes should move towards: evidence specific enough to challenge, with a boundary a buyer can inspect and dispute rather than a confidence claim detached from its method.

Public-sector relevance

The UK State of Digital Government Review reported substantial legacy exposure, fragmented data and critical outages. Public services also carry policy, accessibility, safeguarding, audit and long-lived records. A replacement must preserve these obligations while addressing the identified technical constraints. [State of Digital Government Review](#).

The failed-write rehearsal

During the copy

Update the address and replay changes, including permitted repeat or reorder.

Compare meaning

Identity relationships and effective dates, not only matching record counts.

A measured refactor lets a public body preserve the service while changing the machinery. Domain separation supports policy change. Explicit rules keep consequential answers deterministic. Smaller deployable units reduce the blast radius of change. Owned contracts and evidence reduce dependence on supplier assurance.

Lower implementation cost matters, but avoiding another decade of partial parity matters more.

Worked transition: preserve writes after the switch

Consider a fictional records service whose old system stores one current address per person. The replacement introduces versioned addresses while keeping the existing read contract. Both systems agree on the pre-transition census because every fixture has one current address. A simple traffic switch looks reversible: route back to the old service if errors rise.

After cutover, a user records a future address change. The new representation can hold the current and future versions together; the old representation cannot. Routing back now risks losing the future instruction or overwriting the current address. A flag can reverse traffic routing, but it cannot reverse this semantic difference. The migration plan must settle the post-transition write contract before calling rollback available.

One option is to preserve the old representational boundary during the reversible interval, then enable future-dated changes only after the retirement decision. Another is to implement and test a compatible representation on the old side. A third is to accept a forward-recovery boundary with a rehearsed way to repair the replacement. These choices have different costs and user effects. None should be hidden beneath a generic rollback box on an architecture diagram.

Transition state	Authoritative writer	Evidence needed before advancing
Rehearsal	Old service	Migration reconciles identities, history and rejected records
Shadow comparison	Old service	Replacement observes real-shaped work without emitting live effects
Reversible use	Defined writer for each record	New writes remain representable on the return path; replay is exercised
Expanded representation	Replacement	Authority accepts the changed recovery boundary and new data contract
Retirement	Replacement	Exceptions resolved, recovery demonstrated and old dependencies identified

A concurrency challenge belongs in the rehearsal. Update an address while the initial data copy runs, then replay changes. Confirm that the final current address is neither lost nor overwritten by the older snapshot. Repeat with the same event and with events delivered in an order the chosen transport permits. The comparator should examine the resulting relationships and effective dates, not only matching record counts.

The operator must be able to identify the authoritative writer at every stage. Uncontrolled dual writing creates two competing histories whose disagreement may only appear during recovery. Where dual writing is necessary, specify partial failure, reconciliation and conflict ownership. The buyer should ask to see one failed-write exercise and its repaired business state. A successful demonstration of the ordinary transition is weaker evidence than an exercised account of how the difficult interval ends.

Failure modes

The method can fail if the census is shallow, if normalisation hides material differences, if observed defects are copied without decision, or if the target architecture cannot coexist safely with the old path.

How apparent parity becomes misleading

Over-normalisation

The comparator erases the very difference the obligation cares about.

Copied defect

Both paths agree while preserving a behaviour the owner should correct.

It can also fail commercially. Maintaining two systems costs money. Stakeholders may keep adding scope while demanding strict parity. Suppliers may control interfaces or data extraction. A programme needs authority to decide behaviour and retire paths.

These are reasons to make the boundary explicit at the start. They are not reasons to accept a demo as a replacement.

What buyers should require

Ask a modernisation supplier for:

Ask to inspect the evidence chain

Behaviour

A census with omissions, classifications and compared values, state and effects.

Transition

Rehearsed reconciliation, a recovery boundary and a retirement criterion.

- a map of current domains, contracts and data ownership;
- the behavioural census and its known gaps;
- classification of preserved, corrected and retired behaviour;
- an incremental coexistence and rollback plan;
- differential results covering values, state and effects;
- migration rehearsal and reconciliation evidence;
- operating-cost and performance comparison; and
- an explicit retirement criterion.

“Feature parity” should be an evidence programme, not a future roadmap label.

Conclusion

Documented, observable behaviour provides a basis for rebuilding a system while preserving its agreed capability, provided data access, integration rights and transition constraints permit it.

Recover the truth, separate the domains, build the whole agreed capability, compare both paths and move only when the evidence holds. AI may reduce the mechanical burden; the net benefit must include review and repair. Deterministic contracts, oracles and human decisions keep the replacement honest.

References

- AWS, [Strangler fig pattern](#).
- AWS, [Branch by abstraction](#).
- AWS, [Understanding application decomposition](#).
- Dristi and Dwyer, [A Differential Fuzzing-Based Evaluation of Functional Equivalence in LLM-Generated Code Refactorings](#), 2026.
- UK Government, [State of Digital Government Review](#), 2025.