STRUST v1: An Evidence-Bounded Project Case Study

George Weale
Columbia University (2025)
george.weale@columbia.edu

***Abstract*—This case study asks what the surviving STRUST v1 evidence actually supports. The recovered ten-page project paper contains one ten-row result table, but the executable COBOL/Java pairs, fixtures, raw outputs, comparator policy, and runtime manifest are not preserved. The table records eight *Pass* and two *Fail* rows while the surrounding prose reports 100% accuracy. A hash-bound transcription recovers a document-level rate of $8/10 = 0.80$. A second-cycle evidence graph links five claim classes to ten artifact classes. Under our locally declared graph, 2 of 19 required links survive, and one of five modeled outcome claim classes is fully auditable: recorded table arithmetic. The other four modeled classes remain blocked, and no single missing artifact would restore any one of them. The package is therefore a transparent project postmortem and a protocol for future blinded evaluation, not validated research or a benchmark.**

***Index Terms*—legacy modernization, COBOL, Java, differential testing, evidence audit, reproducibility, project case study**

I. START WITH THE SIMPLEST CHECK

Suppose a detector is described as 100% accurate, but its own result table contains eight passes and two failures. Before debating architecture or statistical significance, the table and the claim must agree. Here they do not. That arithmetic check is the most reliable result preserved for STRUST v1.

The original project idea remains reasonable: run a COBOL reference and a Java candidate on controlled inputs, capture the observations that matter, and compare them. Differential testing is a well-established way to expose disagreement between implementations [2]. But a good idea and a validated implementation are different things. Validation requires the programs, inputs, outputs, decision rule, and execution environment. None of those artifacts survives in the v1 package.

This revision therefore does three things:

1) preserves the historical paper and its table without rewriting the outcomes;
2) derives only quantities that can be traced directly to those rows; and
3) separates useful engineering lessons from claims requiring a new experiment.

The appropriate publication class is **Project case study**. The artifact documents a product hypothesis and an early evaluation record; it does not contain enough evidence to support a research benchmark or a demonstrated quality gate.

II. EVIDENCE SOURCE AND AUDIT METHOD

A. The recovered source

The source is the last pre-reconstruction public PDF in repository history: commit `96dba8008ce8`, path `public/pdfs/Strust.pdf`, Git blob `68ae5dc68371` [1]. Full identifiers are recorded in the machine-readable manifest. A copy is bundled with this case study; its SHA-256 begins `e3626b2d` and ends `dca82881`. The digest establishes which file was audited; it does not establish that the events described by the file occurred.

The complete recovered artifact contains prose, architectural descriptions, pseudocode, one illustrative output diff, and Table I. It does not embed or link a runnable subject repository, fixtures, raw traces, comparator source or configuration, container image digest, compiler flags, or independent label record. Repository history at that revision preserves the PDF but no companion STRUST execution package.

B. A literal, inspectable transformation

The audit uses a deliberately short chain:

1) transcribe Table I, page 7, preserving the historical fields *Expected Result*, *Result*, and *Notes*;
2) rename *Expected Result* to *intended class* so an author-provided label is not mistaken for independent ground truth;
3) encode *Pass* as one and *Fail* as zero; and
4) count the rows and compare their rate with the numerical claim in the source prose.

The analysis script verifies the source PDF hash before reading the ledger. Tests recalculate the counts from the rows, assert that T9 and T10 remain the only recorded failures, and confirm that the data model contains no `ground_truth` field. The package manifest binds the inputs, report, code, tests, result files, figure, and compiled PDF by SHA-256.

III. WHAT THE LEDGER SAYS

Panel (a) of Figure 1 is the complete row-level reconstruction of historical Table I. It contains ten cases: six intended to be equivalent and four intended to be non-equivalent. Eight are marked *Pass*; T9 and T10 are marked *Fail*. The bundled CSV retains every source note and locator. These are source facts, not rerun observations.

The mechanically derived recorded pass rate is $8/10 = 0.80$. This number is often called "accuracy" in the historical document. That word is too strong here. The intended labels and statuses come from the same author, the case-selection process is undocumented, and the executions cannot be checked. *Recorded pass rate* states exactly what was counted.

The source prose nevertheless states 100% accuracy or success in its Results (p. 7), Validation of Feasibility (p. 8), and Conclusion (p. 10). The prose claim therefore exceeds the table-derived rate by $1.00 - 0.80 = 0.20$, or 20 percentage points. Figure 1 shows both the contradiction and the more important evidence boundary.

A. What the two failures can teach

The source attributes T9 to a missed rounding difference and T10 to a false positive on EBCDIC. These are plausible failure modes. Decimal arithmetic can change when fixed-scale COBOL semantics are translated into binary floating-point operations; Java's `BigDecimal` makes scale and rounding explicit [3]. Character data can also disagree at the byte level while representing the same text under different encodings [4].

The evidence stops there. Without the inputs, raw bytes, programs, and comparison policy, the notes cannot show which operation rounded, which code page was used, or whether the comparator rather than the candidate caused the status. They are hypotheses for regression fixtures, not independently verified diagnoses.

IV. What Does Not Follow

A. No independent accuracy estimate

An accuracy estimate requires trusted labels and a defined population. Here, the labels are author intentions in the same table as the statuses, and the ten cases were manually constructed. The artifact provides no sampling frame, blinding, adjudication protocol, or held-out set. The 0.80 rate therefore cannot be generalized to COBOL programs, conversion tools, vendors, or production portfolios.

B. Why this revision reports no p-value

A binomial interval or chance test would be mathematically easy and scientifically distracting. Such calculations assume a probability model for how cases were sampled and repeated. A hand-selected, outcome-known list of ten scenarios does not justify that model, and "50% chance" is not a meaningful operational baseline for a comparator whose behavior is unknown. Reporting a precise p-value would make uncertainty look smaller while leaving the central selection and provenance problems untouched.

Leave-one-case-out arithmetic is equally limited: removing a pass gives $7/9$, and removing a failure gives $8/9$. That merely restates that two of ten rows failed. It does not measure generalization.

C. No feasibility, performance, or scale result

The historical paper describes containerized execution and records one example diff, but a description is not an execution record. No retained evidence shows that all ten pairs compiled, that identical fixtures reached both programs, that outputs were captured before normalization, or that containers were pinned. Although the source mentions timing and memory as secondary metrics, it reports no measurements. Consequently, v1 supplies no empirical support for feasibility, speed, resource use, localization quality, or enterprise scale.

V. Second-Cycle Claim-Evidence Archaeology

The first audit counted rows and missing files. The second asks a sharper counterfactual: which historical claims are inspectable from what survives, and what declared missing evidence bundle would have to be recovered before each blocked modeled claim could be audited? We locally declared one conjunctive requirement set per claim before counting; alternative sufficient bundles were not assessed. The vocabulary follows the official ACM SIGSIM PADS distinction among available, functional, reusable, and results-reproduced artifacts [8], but this local graph is not an ACM badge decision.

Five modeled outcome claim classes are linked to ten artifact classes. A required edge survives only when its artifact is present in the recovered package. Complete coverage under this declared graph makes a claim auditable; it does not make the result favorable.

Under our locally declared graph, the recorded table rate is the one fully auditable class among five modeled outcome claim classes: both of its required artifacts survive. Across those five classes, 2 of 19 required claim-artifact links survive and 17 are missing. The declared missing bundle has six artifacts for independent accuracy, four for technical feasibility, three for execution performance, and four for enterprise scale. The last row preserves the historical negative boundary that scale was not demonstrated.

No single missing artifact completes any blocked modeled claim under this graph. Recovering executable subjects alone, for example, would reduce several declared missing bundles but would not supply fixtures, traces, policy, environment, labels, or measurements. This matters because artifact recovery is not a binary event. A runnable file can still leave the scientific claim unauditable.

The evidence graph therefore strengthens, rather than softens, the project classification. The preserved package can support provenance and table arithmetic. It cannot support independent accuracy, feasibility, performance, or enterprise-scale claims.

VI. Project Lessons Worth Keeping

The audit does not make the project valueless. It changes which lessons are honest to claim.

1) **Comparison is a contract.** Inputs, observation scope, encodings, decimal scale, rounding, ordering, and tolerated variation must be versioned before outcomes are inspected. A permissive normalizer can hide a defect; a narrow normalizer can invent one.
2) **Raw evidence comes first.** Store source and candidate bytes, streams, exit states, and environment identities before canonicalization. Later policy changes can then be audited without erasing the original fact.
3) **Use three outcomes.** MATCH and DIVERGE describe comparable executions. Compilation failure, timeout, missing adapter, or environment mismatch should be INCONCLUSIVE, not forced into a favorable binary score.
4) **The reference is evidence, not truth.** A legacy behavior may be a defect that the Java version intentionally repairs. The oracle problem requires an adjudicated change record rather than silent relabeling [5].

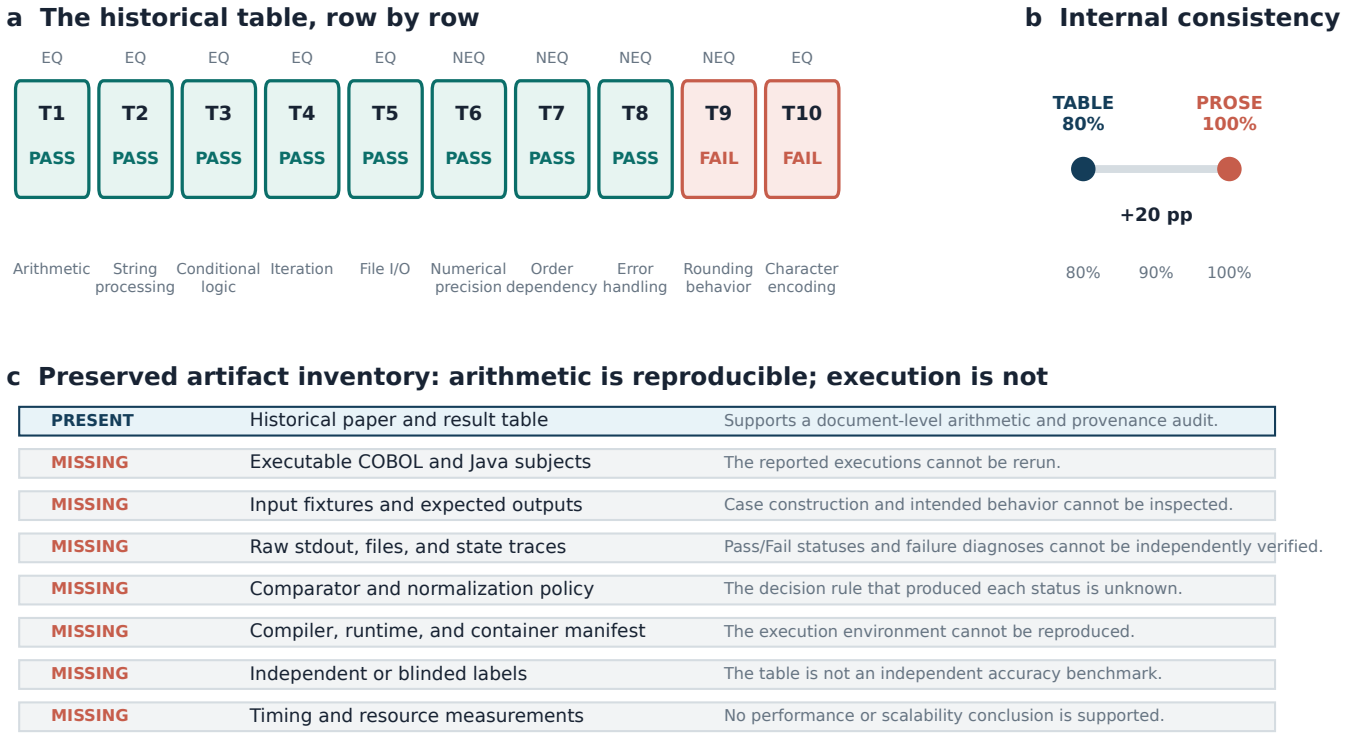

Fig. 1. Evidence audit generated directly from the transcribed ledger and artifact inventory. (a) Every historical case, intended class (EQ/NEQ), and recorded status. (b) The table-derived 80% rate versus the source's 100% prose claim. (c) What survives. Arithmetic is reproducible; the executions, labels, comparator, and environment are not.

These principles are design requirements inferred from the project's failure modes and the testing literature. They remain unvalidated for STRUST itself.

VII. A Study That Could Validate the Idea

The minimum protocol below converts broad ambitions into falsifiable checks. A future study should freeze it before seeing held-out outcomes.

Claim	Evidence and decision rule
Repeatability	Pin subjects, fixtures, commands, toolchains, and raw traces; require a second environment to reproduce each verdict.
Comparator correctness	Use blinded independent labels and a frozen policy; report the full three-way matrix and every error without holdout tuning.
Localization	Use seeded and natural divergences with adjudicated boundaries; predeclare a measure and report misses.
Scale	Exercise multi-program flows and state; report throughput and resource growth with failures and timeouts retained.
Relevance	Define strata across dialects, subsystems, and representations; restrict conclusions to sampled strata and publish exclusions.

Coverage should guide case design but cannot substitute for fault detection or oracle quality [6]. For context-dependent programs, a future verification-slice design may help isolate a named business observation and its dependencies, drawing on program slicing [7]. That is a testable architecture hypothesis, not a v1 result.

VIII. Limitations of This Audit

First, the ledger is manually transcribed from a rendered PDF. The package reduces transcription risk by bundling the exact source, storing page/table locators, testing row identities, and exposing the CSV for review. Second, Git history and a cryptographic digest establish provenance of the audited file, not authenticity of its underlying executions. Third, absence is scoped to the recovered artifact and repository history inspected here; private or lost artifacts may once have existed. If recovered, they should be evaluated as new evidence rather than assumed to validate this reconstruction.

IX. Conclusion

The strongest conclusion from STRUST v1 is simple. Its historical table says eight passes and two failures; its prose says 100%. The reproducible rate is 80%, but even that is a document-level count, not an independent accuracy estimate. Under our locally declared claim-evidence graph, 2 of 19 required links survive and one of five modeled outcome claim classes is fully auditable: recorded table arithmetic. Recovering any one missing artifact would complete none of the four blocked modeled claims. Missing subjects, fixtures, traces, comparator rules, and environment manifests prevent those four classes from being audited from the recovered package. Alternative sufficient bundles were not assessed. Treating the artifact as a project case study makes it more useful, not less. It preserves an honest early result, shows exactly where the evidence chain breaks, and turns two plausible failure notes into requirements for a real follow-up. The next version should earn trust through blinded cases, frozen policies, raw evidence, and independent reproduction.

Fig. 2. Claim-evidence survival. The 19 colored cells are declared required links; only two survive. Artifact codes are A1 historical paper and result table; A2 hash-bound ledger transcription; A3 executable COBOL and Java subjects; A4 fixtures and expected observations; A5 raw outputs and state traces; A6 comparator and normalization policy; A7 compiler, runtime, and container manifest; A8 independent or blinded labels; A9 timing and resource measurements; and A10 declared multi-program workload and sampling frame.

Artifact Availability

The local package contains the historical source PDF, transcribed ledger, configuration, analysis code, tests, generated data, vector and raster figure, paper source and PDF, and a claim/checksum manifest. The analysis requires no network access and performs no historical execution reconstruction.

References

[1] G. Weale, "Strust-v1: Performance Metric for COBOL-to-Java Generative AI," project paper, 2025, repository commit `96dba8008ce831118da87b968dc2b48df79eb373`.

[2] W. M. McKeeman, "Differential testing for software," *Digital Technical Journal*, vol. 10, no. 1, pp. 100–107, 1998.

[3] Oracle, "Class BigDecimal," *Java Platform, Standard Edition 11 API Specification*.

[4] IBM, "Controlling how numeric data is stored," *Enterprise COBOL for z/OS Documentation*.

[5] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[6] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Computing Surveys*, vol. 29, no. 4, pp. 366–427, 1997.

[7] M. Weiser, "Program slicing," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 4, pp. 352–357, 1984.

[8] ACM SIGSIM, *PADS 2026 Reproducibility and Artifact Evaluation*, 2026, accessed Aug. 3, 2026. [Online]. Available: https://sigsim.acm.org/conf/pads/2026/blog/artifact-evaluation/.