International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(21s) 2026 pp. 2086-2098
https://doi.org/10.70917/ijcisim-2026-5956
Article

Regression-Assured Lakehouse Data Products for Multi-Source Supply Chain Transformation

Venu Gopal Kakarla

Independent Researcher, USA

Abstract: A supply chain transformation program rarely converts one system at a time and calls it done. More often, dozens of enterprise resource planning instances, legacy applications, and regional operational platforms feed a single analytics environment, and the reliability of that conversion is treated as an implementation detail rather than an architectural discipline. This article proposes a different framing: lakehouse data product governance, in which curated supply chain objects material, inventory, delivery, order, supplier, and bill-of-materials records are built and maintained as reusable, auditable products, not one-time table builds. The framework layers a raw-curated-aggregated lakehouse pattern with a regression-assurance discipline that keeps validation of actively converting source systems separate from validation of source systems that must stay untouched. Operational observability lineage, job duration, cost, and failure tracking are treated here as defining properties of a trustworthy data product, not a monitoring convenience bolted on afterward. Drawing on architectural patterns observed across enterprise supply chain modernization efforts, the article argues that regression-assured release governance, more than curated storage layers on their own, determines whether a lakehouse data product program holds up as source systems keep changing underneath it. The discussion connects this framework to the existing data mesh and data product literature and makes the case that release-time regression assurance belongs inside the data product's own definition, not alongside it as a separate quality-assurance step.
Keywords: Lakehouse architecture, data product governance, supply chain data integration, regression-assured validation, release governance, operational observability, Medallion architecture, data contracts, lakehouse layering, data quality management

1. Introduction

A material record in one enterprise resource planning instance rarely matches, field for field, the same material as it appears in a separate warehouse management system. An inventory position calculated by one platform will not always agree with a delivery event logged by a regional logistics system that predates it by a decade. None of this is unusual; most large supply chain organizations run on exactly this kind of patchwork, but once those systems are consolidated, whether through acquisition, platform migration, or a deliberate modernization program, the resulting data platform inherits every inconsistency in how each source represented the same underlying business object. Cloud lakehouse architecture is the usual technical answer, and Medallion-style layering into raw, curated, and aggregated zones has become a familiar way to organize it [1], [2]. But lakehouse data product governance is not achieved simply by adopting a layered storage pattern. It depends on something harder to engineer: whether the organization can trust that a curated supply chain object stays correct while the source systems feeding it keep changing.

That distinction matters because supply chain transformation programs are almost never a single migration event. They stretch across years, during which some source systems are being actively rebuilt while others are expected to keep running exactly as before. A regression introduced by a newly converted source can corrupt a downstream metric without anyone noticing until much later, and a regression in a completely unrelated, unconverted source can just as easily be blamed on the conversion effort that had nothing to do with it. Traditional data quality practice built around a one-time validation pass at ingestion was never designed to tell these two failure modes apart, and it offers little guidance on how release governance should behave when both are live possibilities at once.

Copyright: © 2026 by the authors

A concrete scenario illustrates the stakes. A global supply chain organization migrating a regional order-management platform into a shared enterprise resource planning environment will, understandably, focus validation effort on the system being migrated, confirming that new field mappings are correct, that legacy defects are not silently carried forward, and that curated output reconciles against historical reports. A shared transformation library updated to accommodate that migration, however, can just as easily alter behavior for a completely unrelated warehouse management system whose own conversion is not scheduled for another eighteen months. Because no validation process is watching that untouched system for regressions, a resulting discrepancy in a delivery-performance metric may go unnoticed for weeks, and when it eventually surfaces, the organization has no evidence trail connecting the anomaly back to the release that caused it. This is not a hypothetical edge case; it is the ordinary operating condition of a multi-year conversion program, where dozens of source systems occupy every possible combination of converted, converting, and not-yet-scheduled at any given moment.

This article develops a framework this discussion refers to as regression-assured lakehouse data products, treating a curated supply chain object as a governed, versioned, continuously validated product rather than a static table. Three architectural commitments anchor it: a layered lakehouse design separating raw ingestion from curated business logic and aggregated consumption; a regression-assurance discipline that evaluates converting and non-converting source systems on separate validation tracks; and an observability layer that treats lineage, job duration, cost, and failure history as first-class evidence of platform trustworthiness rather than optional operational tooling. What follows situates this framework against existing lakehouse, data mesh, and data quality literature, develops each commitment in turn, and closes by considering what regression-assured governance means in practice for organizations planning multi-source conversion programs.

The stakes of getting this wrong compound the conversion program scales. A ten-source program that treats each conversion as an isolated validation exercise multiplies its blind spots roughly in proportion to the number of curated objects shared across sources, since every release touching shared infrastructure carries the same undetected blast-radius risk regardless of how carefully the converting source itself was tested. Regression-assured governance is offered here not as an incremental refinement to existing lakehouse practice but as the missing discipline that determines whether a data product built on a continuously changing foundation can be trusted at all. The remainder of this article develops that discipline in three stages: first, situating it against existing lakehouse, data mesh, and data-quality literature; then, working through the architectural, contractual, and release-governance mechanics that make regression assurance operational; and finally, considering what this framework implies for how supply chain organizations should plan and budget multi-source conversion programs going forward.

2. Related Work

The lakehouse architecture set out to combine the low-cost, flexible storage of a data lake with the transactional guarantees and schema enforcement long associated with a data warehouse [1]. Making that combination work in practice required a storage layer capable of atomic, consistent updates over files sitting in cloud object storage, a requirement met directly by transactional table formats that add versioning and concurrency control on top of otherwise immutable cloud storage [3]. Together, these two contributions established the technical precondition for treating a lakehouse table as something closer to a governed dataset than a raw file drop. Neither one, though, addresses what belongs inside a curated layer in the first place or how that layer should be validated once it is populated.

Data mesh approaches this gap from an organizational angle rather than a storage-engine one, arguing that data should be treated as a product owned by the domain producing it, discoverable through a self-serve platform, and governed federally rather than centrally [4]. A systematic review of data mesh adoption in practice finds organizations frequently struggle to operationalize "data as a product" beyond the slogan particularly around what a data product's quality guarantee is actually supposed to contain [4]. The framework in this article can be read as one attempt to make that guarantee concrete for a specific domain, supply chain data, by spelling out what regression assurance and observability a curated object needs before it earns the label "product" rather than "table."

Comparative studies of data lake and data warehouse architectures generally converge on a hybrid, layered approach as capturing the strengths of both, but they tend to treat the architecture choice itself as the main design decision, leaving governance and validation as an afterthought [2]. Systematic reviews of data management practice in smart manufacturing supply chains reach a related conclusion from a different angle: most published work describes conceptual frameworks for manufacturing-phase data integration, while validated, production-grade implementations spanning procurement through after-sales service remain the exception [5]. This article sits inside that gap. It does not propose a new storage architecture; it specifies the regression assurance and observability discipline a lakehouse-based supply chain platform needs for its layered design to stay trustworthy in production across the fuller lifecycle that [5] finds under-addressed.

Broader supply chain digital-twin literature makes a parallel argument from the visibility side rather than the storage side. Comprehensive reviews of digital twin adoption in logistics and supply chain systems find that a digital twin's practical value depends on the reliability of the data feeding it, not merely on the sophistication of its simulation logic [6], and cloud computing adoption studies in supply chain management similarly find that cloud infrastructure adoption alone does not guarantee the operational benefits organizations expect without complementary governance practice [7]. Both literatures treat trustworthy data as a precondition their own frameworks assume rather than solve. This article's contribution can be read as supplying exactly that precondition for one specific data domain: the regression-assurance and observability discipline a supply chain data platform needs before the analytics, digital twins, or cloud-native applications built on top of it can be trusted.

A separate, longer-running line of data quality research has established that quality is best understood from the data consumer's vantage point along dimensions like accuracy, completeness, and consistency rather than as a purely technical property of a stored value [8]. Work quantifying what poor data quality actually costs an organization backs this up with numbers: one widely cited case study found that data quality costs at the manufacturing organization studied broke down along both direct-versus-hidden and operational-versus-strategic lines, with the hidden and strategic categories often outweighing the visible, direct ones [9]. More recent empirical work specific to logistics operations pushes this further, finding that inaccurate product master data measurably slowed order-processing throughput at the companies studied [10]. Data warehouse regression testing, meanwhile, has historically been treated as its own discipline apart from general data quality management, with proposed frameworks separating schema-level checks from business-rule checks in ways that anticipate the conversion-scoped validation proposed here [11]. This article pulls these threads together: regression assurance is treated not as a warehouse-testing afterthought but as a defining property of the data product itself, aimed squarely at preventing the hidden, strategic-tier costs identified in the data quality cost literature [9], [10].

A related stream of work addresses data governance and master data management more directly, though largely independent of any single storage architecture. Systematic analysis of data governance frameworks finds that most published frameworks specify governance principles and organizational roles in the abstract, without much guidance on how those principles translate into concrete, machine-enforceable controls at the point where data is actually transformed [12]. Master data management research aimed specifically at supply chain contexts has begun exploring how generative AI techniques might improve entity resolution and record deduplication for shared reference data such as supplier and material records, though this work remains at an early, largely exploratory stage rather than validated production practice [13]. A related line of practitioner-oriented work examines how data governance functions can be better integrated with enterprise resource planning and master data management platforms operationally, arguing that governance succeeds only when it is embedded in day-to-day platform operations rather than administered as a separate compliance layer [14]. Read together, this cluster of work reinforces a pattern already visible in the data mesh literature: governance is widely agreed to matter, but the literature offers comparatively little on how governance principles become enforceable controls inside an actual data pipeline, rather than policy documents that sit beside it. The regression-assurance and data-contract mechanisms developed in Sections 5 and 6 of this article are offered as one concrete answer to that gap, specific to the supply chain domain and the multi-source conversion setting.

3. Fragmented Source Systems and Semantic Risk

The main obstacle to trustworthy supply chain data is rarely a shortage of storage or compute. It is semantic inconsistency across source systems that were never built to agree with each other in the first place. A material identifier in one enterprise resource planning instance may not map one-to-one onto the same physical material as it appears in a regional warehouse system. A delivery status code can carry a different business meaning depending on which logistics platform produced it. A supplier record might be duplicated across systems with subtly different addresses, tax identifiers, or qualification statuses attached. None of this necessarily reflects a defect in any individual source; each system may be doing exactly what it was built to do. The risk only shows up once these sources are consolidated and expected to agree.

Bill-of-materials records add a further layer of difficulty, since the same finished good can be represented by structurally different component hierarchies depending on which manufacturing or engineering system authored the record: a flattened parts list in one legacy system, a multi-level engineering structure in another, and a commercial packaging bundle in a third. None of these representations is wrong for the purpose it was built to serve, but none of them can be compared directly to the others without an explicit reconciliation step that most organizations have never had to formalize, because no single team previously needed all three views reconciled against each other at once. Supplier records raise a related but distinct problem: qualification status, tax identifiers, and banking details often live in whichever system last onboarded that supplier, and a merger or new source-system rollout routinely surfaces two or more supplier records that clearly describe the same legal entity but disagree on nearly every attribute beyond the name.

Real-time supply chain integration platforms compound this fragmentation problem rather than resolving it, since streaming a larger volume of data faster does nothing to reconcile the semantic disagreements already present in the underlying sources; it simply delivers those disagreements to downstream consumers with less delay [15]. Cloud-based enterprise resource planning deployments marketed around real-time visibility face the same limitation: visibility into a fragmented set of business objects is not the same as agreement about what those objects mean [16]. Internet-of-things telemetry adds a further wrinkle beyond timing and identifier conventions: sensor-originated events typically arrive at a far higher frequency and finer granularity than the transactional records they are meant to complement, so a governed data product architecture must decide, deliberately, how a stream of device-level readings aggregates up to match the transactional grain that supply chain planning and reporting already operate on a decision each ad hoc integration effort currently makes independently and often inconsistently, absent a shared object definition to anchor it [17].

Left unresolved, this kind of inconsistency pushes reconciliation work down to individual downstream consumers, and each team ends up inventing its own rules. One reporting group builds a way of resolving material identifiers; another builds a slightly different rule set for the same purpose; the two eventually produce different numbers for what should be a single operational metric. This is a recurring failure mode in real-time supply chain data integration efforts, where the absence of a shared semantic layer has been identified as a primary obstacle to trustworthy cross-system reporting [15], [16]. Internet-of-things-enabled monitoring makes the problem worse still, since sensor-originated events bring their own timing, granularity, and identifier conventions that have to be reconciled against transactional records rather than simply appended alongside them [17].

A governed data product architecture handles this risk by defining common business objects once, at the point where source-system differences are absorbed, rather than repeatedly at the point of consumption. Material, inventory, delivery, order, supplier, and bill-of-materials objects become shared definitions every downstream consumer references instead of reinterpreting them from scratch. That does not make the underlying semantic differences between source systems disappear; those differences are a property of the systems themselves, not something an architecture can wish away, but it does concentrate the burden of resolving them in one auditable place instead of scattering it inconsistently across every team that touches the data afterward. What that single location should actually look like is the question the next section takes up.

4. Lakehouse Data Product Architecture: Raw, Curated, and Aggregated Layers

A layered lakehouse design splits supply chain data across three zones, each with a distinct job. The raw layer keeps source-system data close to its original form minimal transformation, original field names, and even the source's known defects intact so that any downstream issue can be traced back to where it started instead of being obscured by early cleanup. The curated layer applies the business-aligned transformations, naming conventions, deduplication logic, and quality rules that resolve the semantic inconsistencies described above, producing standardized business objects that read the same regardless of which source system originally supplied the underlying record. The aggregated layer builds business-ready metrics and views on top of the curated layer, tailored to whatever reporting, planning, or data science use case needs them, without asking each consumer to repeat the curation logic underneath.

Each layer also implies a different access pattern and a different audience. The raw layer is rarely queried directly by business users; its primary consumers are the curation pipelines themselves and, occasionally, an engineer diagnosing a defect that traces back to source-system behavior. The curated layer is where most governed consumption should happen. Data scientists building a demand-forecasting model, a planning team reconciling inventory positions across regions, and a delivery-performance dashboard should all read from the same curated material, inventory, or delivery object rather than each re-deriving their own version from raw source extracts. The aggregated layer, by contrast, is built for speed and specificity: pre-computed metrics and denormalized views tuned to a particular reporting or planning workload, refreshed on whatever cadence that workload actually needs rather than the cadence the curated layer itself refreshes on. Comparative architectural reviews of data lake and data warehouse approaches to big data analytics converge on this same layered separation as the practical resolution to the lake-versus-warehouse debate, even though those reviews stop short of specifying the governance discipline that determines whether the curated layer they describe can actually be trusted in production [2]. This article treats that governance discipline, not the layering pattern itself, which prior work has already established as the open problem worth addressing.

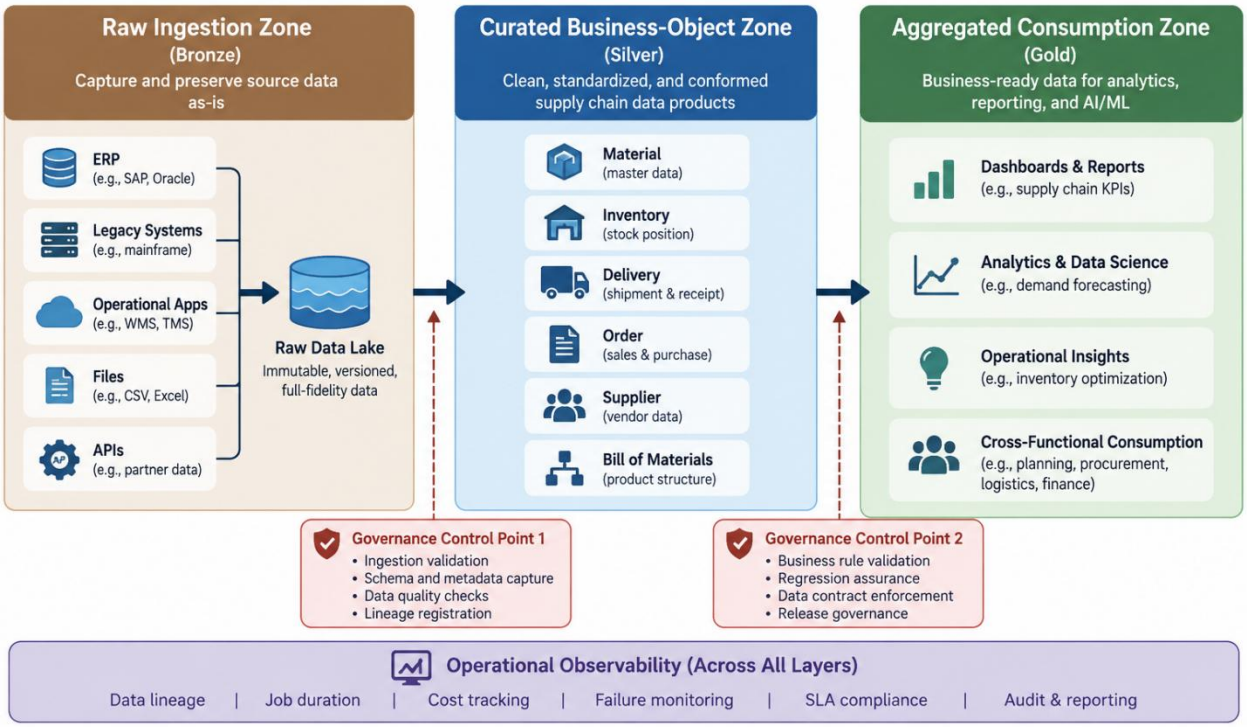


Fig. 1. Lakehouse layering architecture: raw ingestion zone, curated business-object zone, and aggregated consumption zone, annotated with the governance control point at each layer boundary.

Figure 1. Lakehouse layering architecture: raw ingestion zone, curated business-object zone, and aggregated consumption zone, annotated with the governance control point at each layer boundary.

This layering buys two things a flat, undifferentiated data lake cannot offer on its own. First, traceability: because the raw layer is preserved rather than overwritten, a defect discovered in a curated object can be traced back to a specific source-system record instead of forcing a full re-extraction just to diagnose it. Second, reusability: because curation happens once, at the boundary between the raw and curated layers, an executive delivery-performance dashboard, a planning-system feed, and a data science training dataset can all draw on the same curated material or inventory object without each one re-deriving its own version. Transactional table formats built for atomic updates and time travel over cloud object storage make this layering operationally workable at scale they let the curated layer update incrementally as new source data arrives, rather than forcing a full-table rebuild on every refresh [3].

The layered design also scales in a way that matters specifically for multi-source supply chain conversion programs, where new sources are onboarded incrementally over a period of years rather than all at once. Because curation logic is anchored to the boundary between the raw and curated layers rather than embedded inside each individual consuming application, onboarding a new regional enterprise resource planning instance means writing a new raw-to-curated transformation for that source and mapping it onto the existing curated object definition, not rebuilding the aggregated layer, the dashboards, or the downstream data science pipelines that already consume that curated object. Systematic review evidence from smart manufacturing supply chain contexts suggests this incremental-onboarding pattern remains the exception rather than the rule in published implementations, most of which describe integration efforts scoped narrowly to a single manufacturing phase or a single source system rather than an architecture built from the outset to absorb additional sources over a multi-year program [5]. The framework developed here treats that incremental-onboarding capacity as a first-order design requirement, not an incidental benefit: a lakehouse data product architecture that cannot absorb a fifth, tenth, or fifteenth source without re-touching every downstream consumer has not actually solved the fragmentation problem described in Section 3, regardless of how cleanly its first two or three sources were integrated.

Layering by itself, though, says nothing about whether a curated object should be trusted at any given moment. A curated material object can be structurally sound and still reflect logic that has quietly drifted from the business rules its consumers assume it still follows, especially during a stretch when the source systems feeding it are themselves being converted. Closing that gap requires a validation discipline that operates at the point of release, not only at the point of initial curation, which Section 6 takes up directly. Before getting there, though, the architecture needs a way to capture business rules in the first place, since a rule that was never formally written down cannot later be regression-tested against anything.

5. Converting Business Rules into Reusable Data Contracts

Supply chain business logic is often documented, if it is documented at all, in source-to-target mapping spreadsheets maintained by whichever team last touched a given integration. That approach makes the rules hard to audit, easy to duplicate inconsistently across pipelines, and close to impossible to regression-test in any systematic way, since the mapping documents themselves are rarely machine-readable. A more durable pattern converts these mapping documents into explicit, versioned data contracts: structured specifications defining a curated object's expected schema, permissible value ranges, referential relationships to other curated objects, and the transformation rules that produce it from raw source data. Recent proposals for executable data contracts push this further, framing the contract itself as a runnable specification embedded directly in the pipeline rather than a static document consulted separately, and reporting substantial reductions in downstream data defects and pipeline approval cycle time when contracts are enforced this way instead of merely documented [18].

A data contract's value depends as much on its lifecycle discipline as on its initial content. A contract authored once at the start of a source-system conversion and never revisited degrades into exactly the kind of stale documentation it was meant to replace, particularly as business rules evolve over a multi-year program. Treating contracts as versioned artifacts, with each change reviewed, dated, and retrievable in its prior form allows a regression-assurance process to test a new release not only against the current contract but, when a discrepancy surfaces, against the specific contract version in effect at the time a given curated record was produced. This versioning discipline is

what converts a data contract from a one-time specification exercise into a living record capable of supporting root-cause investigation months or years after a given release. Assigning explicit ownership to each contract, typically the domain team closest to the business object the contract describes, further avoids the diffuse-ownership problem that plagued the mapping-spreadsheet approach in the first place, where a document could be edited by whichever engineer most recently touched an integration, with no accountable party positioned to judge whether a proposed change was correct.

TABLE 1. Legacy source-to-target mapping approach compared with a reusable, versioned data-contract approach.

Dimension	Legacy Mapping-Document Approach	Reusable Data-Contract Approach
Format	Spreadsheet or prose mapping document	Structured, versioned specification
Enforcement	Manual review, inconsistently applied	Machine-checkable at pipeline execution
Ownership	Diffuse last engineer to edit the file	Explicitly tied to the curated object's governance owner
Change tracking	Ad hoc: prior versions often overwritten	Versioned; prior contract states retrievable
Regression testability	Requires manual re-interpretation of the mapping document	Directly testable against the contract's own rules

Treating a data contract as reusable rather than pipeline-specific has a direct architectural payoff: the same contract governing how a material object is curated from one enterprise resource planning source can be reused, with source-specific adapters, when a second regional instance of the same platform is converted later in the program. That reuse is what lets a governed data product architecture scale across a multi-year, multi-source conversion program without re-deriving business logic for every new source from scratch. It also turns what would otherwise be tacit institutional knowledge held by whichever engineer happened to build a given pipeline into an explicit, inspectable artifact that survives staff turnover and can be handed directly to a regression-assurance process, the subject of the next section.

The bill-of-materials and supplier reconciliation challenges introduced in Section 3 illustrate why contract reuse matters beyond simple efficiency. A data contract defining the curated bill-of-materials object can specify, once, how a flattened parts list, a multi-level engineering structure, and a commercial packaging bundle are reconciled into a single canonical hierarchy—a reconciliation rule that would otherwise need to be reinvented, inconsistently, by every downstream consumer that touches bill-of-materials data. The same holds for supplier records: a contract specifying how conflicting supplier attributes are resolved when two source records describe the same legal entity turns an ambiguous, judgment-heavy reconciliation problem into an explicit, testable rule that a regression suite can verify held on the most recent release exactly as it held on the release before it. Executable data contract research reports meaningful reductions in downstream defect rates specifically because the contract is enforced automatically at pipeline execution rather than relying on a human reviewer noticing a mismatch after the fact [18]; the framework proposed here extends that same automation-first principle from the individual pipeline to the full multi-source conversion program, where the number of pipelines relying on a shared contract and therefore the cost of a contract violation slipping through grows with every source onboarded.

6. Regression-Assured Release Governance

A large-scale source-system conversion carries a risk that is easy to miss precisely because it does not show up where attention is naturally focused. Engineering effort during a conversion concentrates, reasonably enough, on the source system actively being converted, checking that its new mappings are correct, that its data contract is complete, and that its curated output matches expectations. What is easier to overlook is that the same release can quietly break

curated objects fed by source systems the conversion never touched: a shared transformation library updated for the new source, a reference table repointed incorrectly, or a downstream join altered in a way that ripples into unrelated upstream data. Regression-assured release governance addresses this by splitting validation into two tracks: one that checks the source system currently being converted against its new data contract and a second, independent one that checks every other source system feeding the same curated layer against its prior, unchanged behavior.

Operationalizing this two-track split requires more than a conceptual commitment to testing both categories; it requires the release process to know, mechanically, which curated objects a given release actually touches. In practice this means maintaining a dependency map between source systems, the transformation pipelines that consume them, and the curated objects those pipelines produce—a map that is itself a natural extension of the lineage tracking discussed in Section 7. When a release is proposed, that dependency map determines which curated objects fall on the conversion-specific track (fed, even partially, by the source system being converted) and which fall on the non-converting-source track (fed exclusively by sources unaffected by this particular release). A release touching a shared transformation library used by both a converting and a non-converting source, for instance, would trigger both tracks simultaneously, precisely the scenario that a single-track validation approach, focused only on the source actively being migrated, would miss entirely. This dependency-aware scoping is what allows regression assurance to catch a shared-library regression before it reaches production, rather than relying on a downstream consumer to notice a discrepancy weeks after the fact.

TABLE 2. Source-conversion risks mapped to their corresponding regression-assurance controls.

Source-Conversion Risk	Regression-Assurance Control
New source mapping produces incorrect field values	Contract-conformance checks against the newly authored data contract
Shared transformation logic altered for the new source affects unrelated sources	Independent regression suite re-running prior-release expected outputs for all non-converting sources
Reference or lookup table updated incorrectly during conversion	Referential-integrity checks comparing current lookup values against the last known-good snapshot
Downstream join or aggregation logic altered inadvertently	Aggregated-layer output comparison against pre-release baseline metrics
Release proceeds without an auditable basis for the go/no-go decision	Evidence generation: automated capture of check results, thresholds, and sign-off record for every release

Making this separation work in practice requires a release process that extracts metadata about what a given release actually changes, reads the expected schema and business rules from the relevant data contracts, runs both the conversion-specific checks and the non-converting-source regression checks, and produces an evidence record before promotion. That turns data quality assurance from a discretionary, manually applied review step into a governed release-control mechanism with an auditable trail consistent with proposals for building data-quality controls directly and automatically into pipeline execution, rather than treating them as a gate applied inconsistently between releases [19]. It also reframes the older warehouse-testing distinction between schema-level and business-rule-level checks [11] as a release-governance decision rather than a purely technical testing concern: which checks run against which source, on which release, and who signs off on the resulting evidence before it goes out.

This evidence-generation discipline also resolves a long-standing ambiguity in warehouse-testing practice about where schema-level checks end and business-rule checks begin. Earlier proposals for data warehouse testing strategies treated this as chiefly a technical distinction, with structural checks belonging to one test suite and semantic checks to another, without addressing who should decide which checks apply to which release [11]. Framing that same distinction as a release-governance decision, rather than a purely technical testing concern, clarifies accountability: the data contract, not an ad hoc testing convention, specifies which schema-level and business-rule checks apply to a

given curated object, and the release evidence record documents which of those checks ran, which passed, and who reviewed the result before promotion. This shifts the underlying question from whether the tests passed to which tests were required, whether they were all run, and whether there is a retrievable record of the outcome—a materially higher bar that a purely pass/fail testing mindset does not, on its own, enforce.

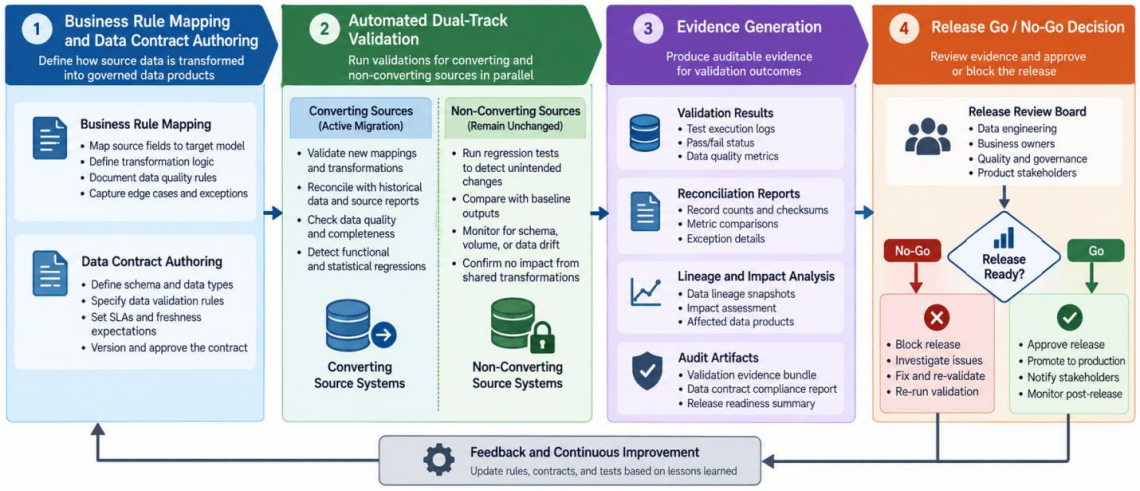


Fig. 2. Release governance flow: business rule mapping and data contract authoring, automated dual-track validation (converting source/non-converting sources), evidence generation, and release go/no-go decision.

Figure 2. Release governance flow: business rule mapping and data contract authoring, automated dual-track validation (converting source/non-converting sources), evidence generation, and release go/no-go decision.

The release-governance flow depicted above also clarifies where human judgment still belongs in an otherwise automated process. Automated checks can confirm that a contract's structural and business rules held, but they cannot, on their own, decide whether an edge case the contract did not anticipate should block a release or be accepted with a documented exception. Reserving that judgment call for a named reviewer rather than either fully automating the go/no-go decision or leaving it to whichever engineer happens to be on call keeps the evidence record honest about where automated assurance ends and human sign-off begins, which matters as much for internal accountability as it does for any external audit of the release process.

The evidence this process generates has a purpose that outlasts the release decision itself. It becomes the audit trail an organization can point to when a downstream consumer questions whether a given metric can be trusted and the diagnostic starting point when a defect eventually turns up since the evidence record shows exactly which checks passed, which were skipped, and under what release conditions. That connects release governance directly to the third architectural commitment of this framework: observability sustained over time, which the next section takes up.

7. Operational Observability: Lineage, Job Duration, Cost, and Failure Tracking

A curated supply chain data product is not finished once its transformation logic is written and its regression checks clear at release time. Unlike a static report, a data product keeps executing on a recurring schedule indefinitely, and how much it can be trusted over time depends on whether the organization can actually see how it is behaving in production, not just whether it nominally succeeded on its last run. Operational observability, in this framework, means systematically capturing lineage (which source records produced which curated output), job duration and resource utilization (whether a pipeline is degrading before it fails outright), cost (whether a transformation's cloud compute spend is proportionate to the value it produces), and failure history (whether a curated object shows a recurring pattern of quality incidents that a single release's evidence record would never reveal on its own).

Cost tracking deserves particular attention because it is the observability signal most often neglected until a cloud infrastructure bill prompts a retroactive investigation. A transformation pipeline whose compute cost per unit

of data processed climbs gradually over several months rarely triggers any alert under a conventional success/failure monitoring setup, since the pipeline continues to complete successfully throughout. Left unexamined, that drift is frequently the first visible symptom of an underlying data quality problem: a join that silently began fanning out more rows than intended, a filter condition that stopped excluding records it was meant to exclude, or a reference table that grew without the pruning logic that once kept it bounded. Treating cost per unit of data volume as an observability signal in its own right, tracked and reviewed on the same cadence as job success rate, surfaces these problems while they are still a cost anomaly rather than after they have become a data quality incident visible to downstream consumers.

Work on end-to-end pipeline observability has shown that combining these signals rather than watching job success or failure in isolation meaningfully improves an organization's ability to catch data quality degradation before it reaches downstream consumers [19]. Executive dashboards that pull this operational signal together alongside business metrics have likewise been shown to shorten the gap between when an anomaly occurs, and when a decision-maker notices it, provided the data feeding those dashboards is governed with the same rigor as the metrics it displays [20]. Without this discipline, an organization risks building a polished consumption layer on top of an operational foundation whose reliability stays invisible until something has already gone wrong downstream. This concern is consistent with recent proposals for comprehensive supply chain data analytics frameworks built specifically to support digital twins, which likewise treat standardized, interoperable, and observable data exchange as a precondition for reliable analytics rather than an implementation detail to be addressed after the fact [21].

Lineage tracking closes a gap the regression-assurance controls from Section 6 cannot close alone. A regression check confirms a given release's output matches expectations at the moment of release; lineage answers a different, later question: why a specific value in an aggregated report looks wrong. Tracing that value back through the curated layer to the exact raw records and transformation logic that produced it, without a full manual investigation, is what turns an isolated release-time control into a lasting governance capability. Job duration and cost tracking do complementary work: a pipeline that is technically succeeding but steadily using more compute time or cost per unit of data volume is showing an early warning sign, often well before that drift turns into an outright quality failure.

Taken together, lineage, performance, cost, and failure-history tracking give an organization something a release-time regression check alone cannot provide: a continuously updated picture of whether a data product's reliability is holding steady, improving, or quietly eroding. That continuous picture is what digital-twin-oriented data analytics frameworks assume is already in place when they describe standardized, interoperable data exchange as a precondition for trustworthy downstream analytics [21], and it is what allows an executive dashboard to reflect not just current business metrics but the governance state of the data underneath them [20]. Without sustained observability, an organization can satisfy every regression check at release time and still have no way of knowing, three months later, whether the data product those checks approved has remained trustworthy in the interim.

8. Discussion: Implications for Enterprise Supply Chain Data Governance

Put together, the three commitments developed above layered lakehouse architecture, regression-assured release governance, and sustained operational observability change what it means for a supply chain data platform to be governed. Existing lakehouse and data mesh work has already established the storage-engine foundation for a governed data platform [1], [3] and the organizational principle that data should be treated as a product rather than a byproduct of pipeline execution [4]. Neither one specifies what a data product's quality guarantee should actually contain once it is running in an environment where source systems are being actively converted underneath it. This article's contribution is to make that guarantee concrete: a supply chain data product earns that label only when its curated output is protected by dual-track regression assurance and made observable through lineage, performance, cost, and failure history, not simply by sitting inside a well-labeled curated lakehouse layer.

That framing has a direct implication for how supply chain organizations should scope multi-source conversion programs. A common failure pattern in practice treats each source-system conversion as an isolated migration project, validating data quality narrowly against that one source and calling it done once its outputs look right. The regression-

assurance discipline in Section 6 argues that this scoping is insufficient: a release's blast radius reaches every curated object sharing infrastructure with the converting source, whether or not those objects were the release's intended target. Organizations planning conversion programs should budget for regression-assurance infrastructure data contracts, dual-track validation, and evidence generation as shared, reusable capability from the very first source-system conversion, rather than as a cost tacked onto each subsequent one. Systematic reviews of smart manufacturing supply chain data practices suggest this reusable-infrastructure pattern is still uncommon, with most published implementations scoped narrowly to a single manufacturing phase rather than spanning a full multi-source program [5] consistent with this article's observation that regression-assurance infrastructure tends to get treated as bespoke to each conversion rather than as shared platform capability.

Adopting this framework also carries organizational implications that go beyond architecture and tooling. Assigning contract ownership to domain teams, as Section 5 recommends, requires those teams to take on a governance responsibility that many currently treat as belonging to a central data or IT function rather than to the business domain itself, a shift consistent with the broader data mesh argument that data ownership should sit with the domain producing the data, not a centralized platform team acting on the domain's behalf [4]. That shift is unlikely to happen automatically alongside a lakehouse migration; it requires deliberate investment in domain teams' data literacy and an explicit reallocation of accountability, neither of which a storage architecture decision addresses on its own. Organizations that treat regression-assured governance purely as a technical capability to be implemented by a platform engineering team, without this accompanying shift in domain ownership, risk building infrastructure that is technically capable of dual-track validation and sustained observability but organizationally unable to use it, because no domain team has been given the authority or the expectation to maintain the data contracts on which the framework depends. This organizational dimension is easy to underweight relative to the architectural and testing mechanics discussed in Sections 4 through 7, but it is arguably the harder of the two problems to solve, since it requires a change in how accountability is distributed across an organization rather than a change in how software is built.

The framework also has something to say about how the cost of data quality failures should be understood inside these programs. Established cost-of-poor-data-quality research separates direct costs from hidden ones and operational costs from strategic ones, generally finding that the hidden and strategic categories dominate total cost even though organizations are least equipped to measure them directly [9]. Regression-assured release governance is, in effect, an attempt to convert some of that hidden cost into a visible, budgeted control for the cost of running dual-track validation and generating release evidence precisely because the alternative, an undetected regression silently corrupting a curated object relied on by executive decision-making, is exactly the kind of hidden, strategic-tier cost this research has found both underestimated and disproportionately damaging.

One limitation worth stating plainly: this framework has been developed at the level of architectural pattern, not validated through a controlled, quantitative before-and-after comparison across multiple independent organizations, and the qualitative case evidence behind it should be read with that scope in mind. Future work extending this framework could usefully quantify the relationship between regression-assurance investment and downstream defect rates across a broader sample of supply chain conversion programs, building on the kind of empirical measurement already emerging in adjacent master-data-quality research [10], [13], [14]. A second open question is how regression-assurance evidence generated at release time should connect to the broader data governance frameworks organizations are adopting independently, for reasons unrelated to any single conversion program [12], an integration this article has not attempted to resolve.

9. Conclusion

Supply chain organizations taking on multi-source data transformation increasingly recognize that a layered lakehouse architecture is necessary infrastructure. This article has argued that layering on its own is not enough to establish lakehouse data product governance. A curated supply chain object becomes a genuine data product, reusable, auditable, and trustworthy for the consumers who depend on it only when it is protected by a regression-assurance discipline that validates converting and non-converting source systems on separate tracks and only when its behavior

in production stays observable through lineage, performance, cost, and failure history rather than visible only at the moment of release. This reframes data quality: not a one-time gate a table passes before publication but a continuous release-governance and observability discipline built into the definition of the data product itself. For supply chain organizations planning large-scale conversion programs, the practical takeaway is to budget for this discipline as shared, reusable infrastructure from the outset, not as an incremental cost tacked onto each source system as it comes up for conversion.

This has practical consequences for how supply chain organizations sequence and budget conversion programs already underway or being planned. Rather than treating regression assurance and observability as capabilities added once problems have already surfaced in production, organizations are better served treating them as part of the initial architectural investment alongside the raw-curated-aggregated layering itself precisely because the cost of retrofitting dual-track validation onto a program with several sources already converted is considerably higher than building it in from the first source onward. The framework presented here has been developed at the level of architectural pattern rather than validated through controlled, quantitative comparison across organizations, and future work quantifying the relationship between regression-assurance investment and downstream defect rates would strengthen the case made here. In the meantime, the central claim stands on its own terms: a lakehouse platform is not a data product architecture merely because it is well laid out in layers, and an organization that wants its curated supply chain objects trusted by the people and systems depending on them needs to treat regression assurance and sustained observability as part of what a data product is, not as work that happens around it.

10. Acknowledgments

The author used Claude (Anthropic, claude-sonnet model) to assist with manuscript drafting and language refinement. The author takes full responsibility for the accuracy, originality, and integrity of all content presented in this manuscript and confirms that all data, analysis, and conclusions reflect independent work and verified sources.

Funding

This research received no external funding.

Author Contribution Statement

Venugopal Kakarla: Conceptualization, Methodology, Investigation, Writing – Original Draft, Writing – Review & Editing, Visualization.

Conflict of Interest Statement

The author declares no conflict of interest.

Data Availability Statement

This article draws on generalized architectural patterns from professional practice. No proprietary, confidential, or client-identifying data are disclosed, and no new datasets were generated for this article.

REFERENCES

1. M. Zaharia, A. Ghodsi, R. Xin, and M. Armbrust, "Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics," in Proc. CIDR, 2021.
2. S. Mezzoudj, M. Khelifa, and Y. Saadna, "Data Lakes versus Data Warehouses: Choosing the Right Approach for Big Data Analytics," J. Electr. Syst. Inf. Technol., vol. 12, no. 1, art. 89, 2025.
3. M. Armbrust et al., "Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores," Proc. VLDB Endowment, vol. 13, no. 12, pp. 3411–3424, 2020.
4. A. Goedegebuure, I. Kumara, S. Driessen, D. Di Nucci, G. Monsieur, W.-J. van den Heuvel, and D. A. Tamburri, "Data Mesh: A Systematic Gray Literature Review," ACM Comput. Surv., 2024.
5. N. Smina, Y. Gahi, and J. Gharib, "Data Management in Smart Manufacturing Supply Chains: A Systematic Review of Practices and Applications (2020–2025)," Information, vol. 17, no. 1, art. 19, 2026.
6. T. Le and R. Fan, "Digital Twins for Logistics and Supply Chain Systems: Literature Review, Conceptual Framework, Research Potential, and Practical Challenges," Comput. Ind. Eng., vol. 187, art. 109768, 2023.

7. M. Yenugula, S. K. Sahoo, and S. S. Goswami, "Cloud Computing in Supply Chain Management: Exploring the Relationship," Manage. Sci. Lett., vol. 13, pp. 193–210, 2023.
8. R. Y. Wang and D. M. Strong, "Beyond Accuracy: What Data Quality Means to Data Consumers," J. Manage. Inf. Syst., vol. 12, no. 4, pp. 5–33, 1996.
9. A. Haug, F. Zachariassen, and D. van Liempd, "The Costs of Poor Data Quality," J. Ind. Eng. Manage., vol. 4, no. 2, pp. 168–193, 2011.
10. D. Božić, M. Živičnjak, R. Stanković, and A. Ignjatić, "Impact of the Product Master Data Quality on the Logistics Process Performance," Logistics, vol. 8, no. 2, art. 43, 2024.
11. D. A. Hamed, N. ElGamal, and E. Hassanein, "A Data Warehouse Testing Strategy for Banks," Int. J. Recent Technol. Eng., vol. 8, no. 4, pp. 8978–8986, 2019.
12. D. Acev, S. Biyani, F. Rieder, T. T. Aldenhoff, M. Blazevic, D. M. Riehle, and M. A. Wimmer, "Systematic Analysis of Data Governance Frameworks and Their Relevance to Data Trusts," Manage. Rev. Q., pp. 1–54, 2025.
13. J. Shah and A. Singh, "Master Data Management for Global Supply Chains: Enhancing Data Quality and Governance with Gen AI," Int. J. Creative Res. Thoughts, vol. 13, no. 2, pp. i555–i563, 2025.
14. C. Bonthu, "The Role of Data Governance in Strengthening ERP and MDM Collaboration," Int. J. Comput. Exp. Sci. Eng., vol. 11, no. 3, 2025.
15. L. B. Mullapudi, "Optimizing Supply Chain Management with Real-Time Data Integration Platform," Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol., vol. 11, no. 1, pp. 442–453, 2025.
16. R. K. Khokrale, "Using Cloud-Based ERP Solutions for Real-Time Supply Chain Visibility and Decision-Making," Int. J. Res. Publ. Eng. Technol. Manage., vol. 7, no. 2, pp. 9811–9830, 2024.
17. Md. R. Karim, C. Rodgers, and Md. A. Hossain, "The Role of Internet of Things (IoT) in Real-Time Supply Chain Monitoring," Int. J. Res. Innov. Soc. Sci., vol. 8, no. 10, pp. 1800–1816, 2024.
18. S. Gurram, "Executable Data Contracts for Reliable AI Pipelines," Int. J. Intell. Syst. Appl. Eng., vol. 13, no. 1, pp. 504–513, 2025.
19. T. T. Bukhari, O. Oladimeji, E. D. Etim, and J. O. Ajayi, "Advances in End-to-End Pipeline Observability for Data Quality Assurance in Complex Analytics Systems," Int. J. Adv. Multidiscip. Res. Stud., vol. 4, no. 4, pp. 1465–1487, 2024.
20. O. T. Kufile, B. O. Otokiti, A. Y. Onifade, B. Ogunwale, and C. H. Okolo, "Building Executive Dashboards for Real-Time Cross-Channel Performance Monitoring," Int. J. Sci. Res. Humanit. Soc. Sci., vol. 1, no. 2, pp. 143–160, 2024.
21. V. Xiros, J. M. Gonzalez Castro, F. Fernandez-Pelaez, B. Magoutas, and K. Christidis, "Supply Chain Data Analytics for Digital Twins: A Comprehensive Framework," Appl. Sci., vol. 15, no. 12, art. 6939, 2025.