

THE DATA CONTRACT THAT SAVED MY PIPELINES: ENFORCING THE OPEN DATA CONTRACT STANDARD FOR RELIABLE CROSS-TEAM DATA PIPELINES

Santosh Nagnath Mehtre

Independent Researcher, USA

Abstract: This study discusses the failure mode where the pipeline's successful completion and no meaningful output is produced due to schema modifications at the upstream that are not poised to be detected by a downstream system. The structural issue is that it is architecturally flawed in decentralized data platforms more generally, the agreement between a producing team with its consumers teams who consume that data is encoded in a place where a machine can verify. The paper suggests integrating that rather informal contract by a formal, versioned, machine-readable document, and considers the Open Data Contract Standard (ODCS), the YAML document standard under the management of the Bitol Project of the Linux Foundation, as the means. It is a typical ODCS v3 contract for a daily orders dataset that is designed and annotated that tends to combine schema, data quality rules such as domain business logic, and service-level properties and ownership into a single document to be reviewed. The following two-gate enforcement architecture is then discussed: structural gate running in continuous integration at the pull-request time, and behavioral gate running on the same data between transformation and publication that is materialized. Experience with adoption of production changes suggests that an incidence of breaking-change moves to producer build-time and that dataset onboarding effort declines, and that quality responsibility moves to the producing realm. Boundary situations where contract overhead cannot be justified are determined and a minimum adoption way has been suggested.

Keywords: Data Contracts, ODCS, Bitol, Schema Evolution, Data Quality, Data Mesh, Pipeline Reliability, Data Governance, Continuous Integration

1. INTRODUCTION

Background

A data contract is a formal, versioned, machine readable description of a dataset that contains, its meaning, its quality guarantees and is responsible for that is agreed upon by the teams that produce the data and those that consume it. This concept has been needed very recently, and why this is so, is due to architecture [29]. Schema coordination is free in a monolithic warehouse owned by one team, since the team, which does the change, is the team, which makes it right; the cost and knowledge needed to analyze the change is in one place. That set-up is disbanded through decentralization. The relocation of the ownership of analytical data to an outer domain of teams creating it, in accordance with the laws of domain ownership, data as a product, self-serve platform and federated computational governance [1]. The producer and the consumer of a dataset do not become the same people, do not share a publish cycle and they often do not share a time zone [2]. The principle of data as product suggests that a dataset has an interface and an interface suggests a contract. This has been long since addressed in software engineering as application programming interfaces are published in schemas, have its changes versioned under an explicit compatibility policy [3]. Incompatible changes are rejected by the build instead of being detected by the caller.



Analytical data do not have an equivalent mechanism. Schemas are stored in wikis, whatever transformation job changed to the point of embedding quality expectations, freshness guarantees survived as tribal knowledge, and chat handles are represented as ownership, which became stale the moment that person changed teams.

Research Aim

This study aims to design and test an enforcement architecture based on contracts, to implement the producer and consumer agreement in a set of analytical data in machine readable form as a specification, and to check that specification automatically both at change time and at run time.

Objectives

- *To investigate the origins of specifications of machine-readable data contracts along with their governance and structure, specifically the Open Data Contract Standard.*
- *To measure the influence of upstream schema evolution of a pipeline on the reliability of downstream pipelines, and to characterize the class of failures that is not detected by standard pipeline monitoring.*
- *To identify the issues of transitioning to contract-based governance within independently operating domain teams.*
- *To propose an enforcement architecture and a minimal adoption path that minimizes breaking-change events and a contract overhead where it is not warranted.*

Problem Statement

Most data pipelines fail most expensively when they complete successfully against data that has suddenly changed meaning rather than when they crash or malfunction. The inspiring event behind this research is an ordinary one as opposed to a special one. The empty dashboards are executive-facing finance dashboards as opposed to the wrong ones [30]. The walk backward through the pipeline revealed the table being served is empty, and that the transformation job had finished indicating that no error had been raised. In the raw landing area, a column on which the pipeline relied, `order_amount` - had been renamed to `order_amount_usd` by the producing team the previous evening. The change had been tried by that team and their tests are successful, as their tests only detailed their own world.

None of the participants are careless. The producer had a defensible modification to an internal detail that the producer saw as a column and had no way of finding out that three internal groups had developed load-bearing logic based on its name. The joins matched nothing, the aggregation added nothing and the successful job wrote a structurally correct table with no rows. The pipeline failed not to perform the actions given, however to do so with input data that no longer has the same meaning as the actions expected.

This type of failure can be challenging to monitor due to two properties of this category of failures. First, it is silent such as orchestration level status, row count free assertions and job level alerting is all healthy. Second, its price is inefficiently distributed, such as the producer replacing a column and moving on, as oblivious as he/she is innocent, when breakage turns up hours down the line in the pipeline of a consumer. The least vulnerable on the way to the fault are the participants who are in the best position to stop the fault. It is that asymmetry, and not any one fault, which this present study attempts to deal with. Experimental studies of schema evolution have confirmed the non-rhetoric and non-incidental nature of these changes: schema evolutions are extremely common throughout the lifetime of database programs, and a significant fraction of those that do occur impact existing information and the code that depends on it [4], [5].

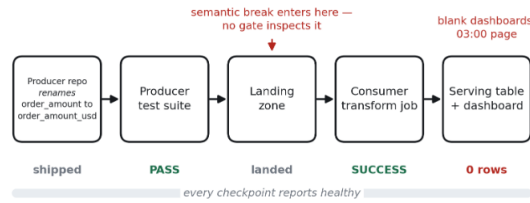


Fig. 1: The silent failure path. Every checkpoint report success while the semantic break propagates undetected

Novel Contribution

Existing treatments of the constituent concerns separately. Schema management is considered a database evolution problem [5], [6]; data quality is considered a problem of validation that is tackled by test suites executed after the fact [7], [8]; governance and ownership is considered a problem of an organization [2], [9]. By merging the four areas of concern, namely schema, quality including domain business rules, service levels and ownership of the procedural act, this study describes the combination of the four areas of concern into one versioned artifact that is expressed in an open, community governed specification [10] and shows that the same artifact can be executed at two points of enforcement to dislocation failure detection to consumer run time, to producer change time.

Fig. 2: Four concerns previously distributed across four artifacts, consolidated into one versioned document

2. LITERATURE REVIEW

Examine the origins, governance and structure of machine-readable data contracts

The industrial heritage of data contracts standardization can be traced back. In May 2023, PayPal, which has adopted one as a data mesh, published its YAML data contract template under an Apache 2.0 license as open source [11], [12]. As of September 2023, the template has been donated to the Linux Foundation and called the Open Data Contract Standard; it is currently governed by Bitol, a project of the LF AI & Data Foundation established in November 2023 [13]. The version 3 of the specification re-organized the fundamentals, schema and data quality sections as well as introduced special support and communication channels, with the version 3.1.0, in December 2025, added relationships, more detailed metadata and stricter validation and full backward compatibility of v3.0.x documents [10], [14].

The model of governance is relevant to adoption, not an incidental aspect of adoption. An encoding of organizational agreements can safely be encoded on a neutral basis, where it is reviewed by the community and with a published compatibility policy, but is not when the encoding is a single vendor encoding format such as it necessarily involves a migration liability at the most established point in the agreement [13].

Structurally, an ODCS document brings together four concerns which traditional practice considers being spread over four places or not at all. Schema block defines logical and physical type of table and columns of a table, null mode and keys. The quality block binds expectations to the schema that can be in the form of library rules of common cases or SQL predicates of domain-specific logic. The frequency, latency and retention are declared in the service-level block. The domain is called the governance block, the responsible owner and the support channel [10].

Evaluate the effect of schema evolution on downstream pipeline reliability

Empirical research on schema evolution is not a recent development, and it seems that research findings are aligned. Much of the early longitudinal evidence observed schema change at a very high rate in the lifetime of a production system, with a large proportion of the changes having an impact on existing data, necessitating migration [5]. Co-evolution Analysis of co-evolution of schema and application code showed that schema changes will be reflected in the dependent code and that a large fraction of schema changes do not include concomitant changes in the

code contained in the same committed transaction [4]. A large public schema longitudinally examined types of change, and measured the impact of change on dependent queries, demonstrating a quantifiable loss in query success with each version [6]. The field surveys confirm that the problem of schema evolution has not been addressed yet in databases as well as data warehouse [5].

The mechanisms proposed in that literature are largely internal to the database

The operators of modification and the rewritten queries, and the migration tooling is automated. Less explicitly what literature speaks about is the cross-organizational case, in which the code that the schema owner is dependent on is that of a team that the schema owner is not able to list. In this configuration migration tooling is inadequate, as the issue is not a query translation but the realization of the query existing. Semantic versioning provides the missing convention by providing a compatibility promise to a version identifier, to enable an incompatible change to be declared and hence detectable [3].

Assess automated data quality verification as an enforcement mechanism

The conceptual foundation of considering quality as multi-dimensional, as opposed to binary that was developed by survey research that revealed that data consumers have significantly different conceptualization of quality than the information systems professionals do, and that the dimensions are grouped into the categories of intrinsic, contextual, representational and accessibility category [15]. Accuracy, completeness, consistency, validity, uniqueness and timeliness among others are thereafter combined in a working taxonomy consolidating the Practitioner frameworks [16]. Tagging each declared rule with their dimension reduces an unordered set of claims to a reviewable quality posture, which is precisely the feature demanded of a contract that can be pulled-request reviewed.

On the implementation side, massive automated checking has been shown to be a realistic production attribute, and declarative constraint definition transcoded into effective aggregate queries on huge volumes of information [8]. Parallel machine learning Work Parallel, schema-based validation is introduced as a training-serving skew and upstream data error-detecting mechanism, and shown to need explicit treatment of data as a production asset with the same rigor as code, should be treated as such [7]. Some recent survey research documents the evolution of data quality awareness out of the traditional data management and to data science systems and observes the change of post-hoc detection to declarative, pipeline-based verification [17]. Taken together this literature provides the execution substrate upon which a quality block of a contract is based; the contribution provided by the contract is to make those declarations, previously spread across transformation code, into a single artifact which is owned by the contract.

Identify challenges in adopting contract-based governance across domain teams

Adoption challenges are both an organizational and technical challenge. The systematic review of the data mesh practice reveals that the least developed of four principles are applied in industrial settings as federated computational governance, and the balance between domain autonomy and a global consistency is often one of the challenges to recycle [2]. Study of decentralized governance systems observes that policy when represented in documentation as opposed to existence of executable artifacts debases predictably, and that it is the capacity to enforce that sets the method [9]. Current qualitative research on operationalization in machine learning states that the engineering embeds defensive validation logic, regularly, down the value chain since its engineers cannot depend on guarantees along the up-stream, and pay the same costs of work per-consumer of a dataset [18].

The second issue is related to scope discipline. Contract specification is an overhead on authoring and review, warranted at inter-team interfaces, but not warranted at datasets used as input to financial, regulatory or decision-critical processes, or at datasets used as a means of exploration under a single team ownership. This decision is not yet represented in the literature in a quantitative threshold and it is a decision based on the qualitative one.

Literature Gap

The existing literature on data is quite extensive, covering data contracts, schema evolution, data quality checking and governance processes, but there is little literature on the exploration of machine-readable data contracts

as a unified enforcement scheme at distributed domain groups. The absence of empirical testing on the issues of adoption, improvement in reliability, efficiency of governance and scalability leaves a big gap to explore further.

3. METHODOLOGY

Research Design

This research does not use a control experiment, but its design is adopting a design and deployment approach. A production-shaped dataset is specified as a representative contract, this is designed and production operation is observed. The analysis is qualitative and descriptive; there is no controlled comparison with a control set up and the results that have been presented in Section IV must be interpreted in this manner. The unit of analysis is a single dataset interface among a team that produces more than one team that consumes. With source control and continuous integration hosted on GitHub [23], the reference stack includes Apache Airflow to orchestrate its other components, including Dataproc and Python Spark to transform data, [20], [21], and Big Query to store and serve the data [22].

Contract Specification

Contract Specification ODCS v3 contains the expression of the contract, which is stored in the Git repository that includes the code that regulates the dataset, not in one of the runtime systems [10]. By putting the specification alongside the code, it is what makes it readable in a pull request and in effect, enforceable at the time of change. An example contract of a daily orders data is reduced to be readable; each block is valid ODCS v3.

```
apiVersion: v3.0.0
kind: DataContract
id: orders-daily-finance
name: Orders Daily
version: 1.2.0
status: active
domain: finance
tenant: RetailCorp
description:
  purpose: Cleaned daily orders feeding revenue and finance reporting.
  usage: Source of truth for net revenue. Do not join on customer_email.
  limitations: Excludes test orders and internal transfers.
```

```
schema:
- name: orders
  physicalType: table
  physicalName: orders_daily_v1
  properties:
    - name: order_id
      logicalType: string
      physicalType: varchar(36)
      required: true
      unique: true
      primaryKey: true
    - name: order_date
      logicalType: date
      required: true
    - name: gross_amount
      logicalType: number
      required: true
    - name: discount_amount
      logicalType: number
      required: true
    - name: net_amount
```

logicalType: number required: true - name: currency_code logicalType: string physicalType: char(3) required: true quality: <i># COMPLETENESS - the data that must be there, is there</i> - type: library rule: nullCheck dimension: completeness description: Core financial fields are never null. properties: [order_id, gross_amount, net_amount] <i># ACCURACY - values fall within a meaningful, valid range</i> - type: sql dimension: accuracy description: Currency must be a supported ISO 4217 code. query: > SELECT COUNT(*) FROM orders WHERE currency_code NOT IN ('USD','EUR','GBP') mustBe: 0 <i># CONSISTENCY - the business rule holds across fields</i> - type: sql dimension: consistency description: Net equals gross minus discount, to the cent. query: > SELECT COUNT(*) FROM orders WHERE ABS(net_amount - (gross_amount - discount_amount)) > 0.01 mustBe: 0 slaProperties: - property: frequency value: 1 unit: d - property: latency value: 4 unit: h - property: retention value: 7 unit: y team: - username: priya.s role: owner support: - channel: '#orders-data' tool: slack url: https://retailcorp.slack.com/orders-data
--

TABLE 1: CONTRACT BLOCKS AND THE CONCERN EACH ENCODES

Block	Concern encoded	Prior location in conventional practice
Fundamentals	Identity, domain, version, status	Wiki page or dataset naming convention
schema	Structure, types, nullability, keys	Warehouse catalogue, inferred at read time

quality	Completeness, accuracy, consistency rules	Scattered transformation and test code
slaProperties	Frequency, latency, retention	Tribal knowledge, undocumented
team / support	Accountability and escalation path	Chat handle, frequently stale

The fundamentals state identity, domain and version, the latter as a semantic versioning statement to ensure the major number has a promise of compatibility, as opposed to a decorative increment [3]. Order_id is defined and made a unique primary key, mandatory in the schema block, where the renaming incident as outlined in Section I re-occurs; with this starting again to not match the reality, the contract exits the world in as ugly a way as possible.

Quality Rule Design

Rules in the quality block are marked in which dimension they pertain, based upon the existing taxonomy [15], [16]. There are three facets that are exercised in the reference contract. Complete states the fact that there is never a field of core finance that is null. Accuracy asserts Each currency_code is an ISO 4217 code [24], and the constraining values do not simply exist, but have a value in the meaningful range. Consistency represents the business rule as such, net amount must be equal to the gross amount minus discount amount to the cent.

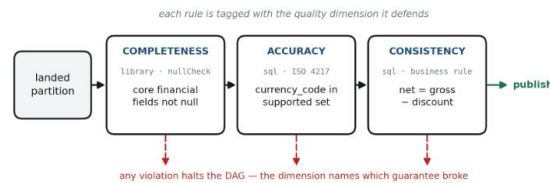


Fig. 3: The quality block as an ordered gate. Each rule carries the dimension it defends

The consistency rule demonstrates influence of the contract-based specification that can significantly contribute. The rule is traditionally stored within a transformation model that had been written usually by an engineer that is no longer in the team, and which is never read by any documentation [25]. It is declared in the contract, marked with its dimension which can be viewed by the consumers, and can be reviewed during a pull request. ODCS allows library rules to be applied to common cases, along with SQL predicates to represent business-specific logic, and that is the reason why it allows business rules and structural constraints to reside in a single artifact [10].

Enforcement Architecture

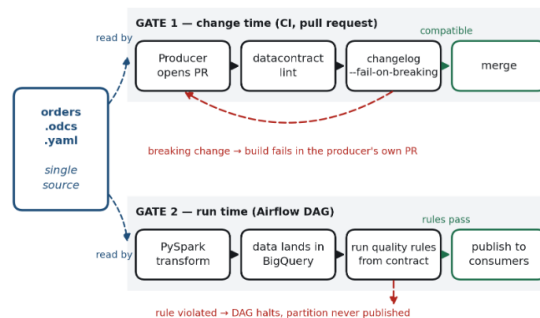


Fig. 4: Contract residence and the two enforcement gates

Contract Residence and the Two Enforcement Gates show the efficiency of data contracts represented in machine-readable formats which are guaranteed to be reliable by pre-deployment and real-time testing. Gate1 blocks incompatible schema changes during development, whereas Gate2 implements quality during published data to the consumers.

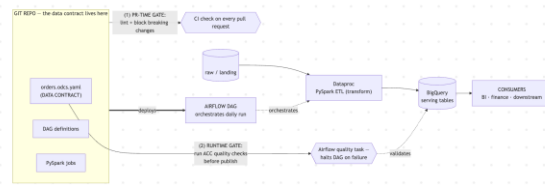


Fig. 5: Enforced architecture

A contract in a repository which is not executed becomes documentation, and decays as documentation decays. The change to enforcement is when the artifact is read by a machine each time it changes and each time it runs. There are two gates that are defined. Gate 1 Structural validation at pull-request time. As a producer is submitting a pull request to the dataset, continuous integration [23] checks the contract on two axes: whether this document is still valid ODCS, and whether the change is a breaking test that breaks the version that consumers rely on. Renaming, dropping, tightening a type with an existing 1.x contract, is a breaking change under the semantic versioning [3], and the build fails, in the same pull request of the producer. The price of the change is consequently shifted to where the decision is decided.

```
# .github/workflows/data-contract.yml
name: data-contract
on: [pull_request]
jobs:
  validate:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Lint the ODCS contract
        run: datacontract lint orders.odcs.yaml
      - name: Block breaking schema changes
        run: datacontract changelog orders.odcs.yaml —fail-on-breaking
```

Gate 2 — Behavioral validation against materialized data

The quality block is functional as opposed to a description block. The announced regulations execute by the use of tests within the pipeline following the data landing and preceding the publication downstream in the form of a task of Airflow (between the PySpark transformation and the publication step) [19], [21]. Checks of completeness, accuracy and consistency run against the serving table in the BigQuery [22]; and the moment net_amount no longer equals gross amount less discount amount then the DAG stops and the corresponding partition to a dashboard is not achieved. Implementation is given to a quality engine - ODCS is read in Soda and Great Expectations are engines of checks [26], [27], [28] almost engine-neutral implementation - the contract is the sole input and both gates read it.

Evaluation Strategy

Measures are evaluated based on the four criteria based on the goals; the breaking point at which the break-even point to detect changes; the effort needed on-the-part of a new consumer to declare datasets correctness; the point-of-accountability in case of a quality defect; and availability of explicit, versioned response to the query of whether one can count on a dataset. Evaluation is an exercise of observation and based on production operation. Instrumented measurement was not done quantitatively in terms of rate of incidence being a limitation mentioned in Section IV.

4. RESULT AND DISCUSSION

Observed Outcomes

Migration of breaking-change detection from run time to change time

The producer generated breaking changes in the pull request which failed a continuous integration test instead of a zero-row table appearing in the pipeline of a consumer. This study is prompted by the incident in which the incident category was production outage which escalated to failed build. The enhancement is not improved producer diligence, merely that the architecture no longer relied on producer diligence which explains the argument that it is the use of computational enforcement as opposed to documented policy that defines the differentiating governance society [9].

Reduction in dataset onboarding effort

The issue of what a dataset contains and whether or not it can be trusted is resolved by reference to the contract as opposed to reference to a wiki page and a named individual. The contract is not out-of-date due to its construction since a contract that distorts the data makes the pipeline leak. This helps in overcoming the currency issue experienced with documentation methods.

Relocation of quality accountability to the producing domain

Rules residents in the producer owned contract transferred responsibility to the producing arena, and downstream defensive cleanup code is put to rest. This is identical to the duplicated-defensive-effort pattern observed in qualitative research of production data/ practising machine learning [18], and is the accountability implication of the data-as-a-product principle [1].

Availability of an explicit dependability answer

Whether a dataset could be extended on no longer is a matter of personal opinion however one with a versioned and imposed response. The author considers this outcome, and not a single incident that has been prevented, to be the main engineering posture shift; there is now dependency on inputs as opposed to defense.

TABLE 2: SUMMARY OF OBSERVED OUTCOMES

Criterion	Before contract enforcement	After contract enforcement
Breaking-change detection point	Consumer pipeline, at run time	Producer pull request, at change time
Failure signal	Silent; job reports success	Explicit; build fails, attributed
Dataset onboarding reference	Wiki page plus named individual	Contract, current by construction
Quality accountability	Consumer, via defensive logic	Producer, via owned contract rules
Dependability determination	Individual judgement	Versioned, enforced answer

Production operation qualitative observations are outcomes. None of the controlled baseline comparison or quantitative incident-rate instrumentation is done.

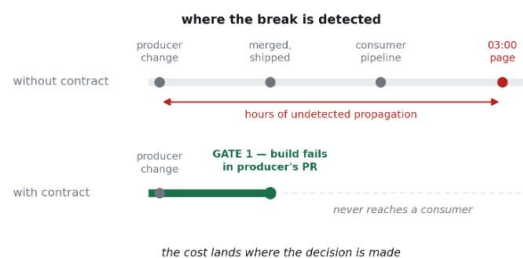


Fig. 6: Relocation of the detection point from consumer run time to producer change time

Discussion

The findings show that the qualitative factor that decides matters is not the specification expressiveness, however the presence of an enforcement location in advance of the change ships. A document which is written, checked and not signed restates the failure mode of the document it is written to replace. The two gates are thus warranted and targeted towards complementary failures classes such as, the structural gate takes care of alterations to the form of the data, and the behavioral gate to the content. Neither concept subsumes the other as a schema-conformant data set may nonetheless violate a domain business rule and a data set that obeys all the quality rules may nonetheless be groomed out under its consumers.

One of the secondary observations is related to artifact economy. The YAML document serves three functions simultaneously: it is the documentation accessed by a new engineer, the schema gate that eliminates breaking changes at the source and the test suite that verifies the data on every run. Traditional practice has three distinct artifacts to these ends, which are separate in the absence of active care. The drifting is eliminated by construction through the process of consolidation.

Boundary conditions

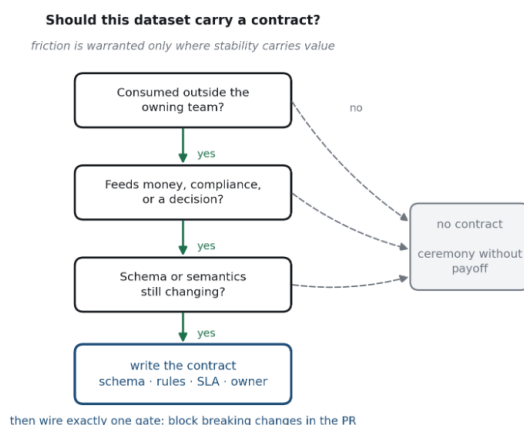


Fig. 7: Scope criterion for contract adoption

Contracts are frictional and the only friction that is justified is one which has value attached to it like stability. In the case of an exploratory dataset or a table which is queried and owned by one analyst end to end, full specification of the contract is overhead with no associated payback. The field makes itself costly at boundaries such as interfaces between teams, and data source or data sink feeding financial, regulatory or decision-critical processes, that has a busted assumption is a courageous cost. This scope criterion is also qualitative and the formalization of the same is referred to as work to be done in future.

Limitations

The assessment is an observation and single-context. There has been no controlled comparison with a baseline environment, or instrumentation of incident rates prior to adoption and after adoption and the results, thus, are directional however not quantified. This study lacks generalization over the different fully formed cloud data platforms.

6. CONCLUSION AND FUTURE RESEARCH

Conclusion

The renamed column never causes the substantive challenge. The substantive issue is that not a single aspect of the architecture captured the agreement between the team that 3D-printed the data, and the teams that used it. A handshake even being carefully followed is neither enforced nor remembered, nor can be scaled to the same level that decentralized data ownership exists.

The paper has demonstrated that by putting that agreement in a versioned, machine-readable document of ODCS, and enforcing it at two different enforcement points, this shifts the breaking-change identification to producer change time, achieving four separately reviewed concerns in one document, and placing quality responsibility back with the owner of the data. The suggested course of action to start with is deliberately low-key such as picks the worst interface that an organization has at present, signed one ODCS contract on it including schema, few completeness, accuracy and consistency constraints, a service-level statement and a named owner, and connect to it with exactly one gate to ensure that breaking changes in the interface of a producer are blocked in the pull request. It only takes one gate to alter the conditions of the discourse.

The renamed column can recur. The value of a contract is to determine whether it is discovered that some contribution is in production at three in the morning or in a pull request at three in the afternoon.

Future Scope

In future investigations, the quantitative gap in Section IV, can be quantified by measuring incident rates, mean time to detect and change-failure rates with adoption of the contract in place to determine measured instead of observed effect. There is a second direction of formalizing the scope criterion, creating a quantitative threshold, based on the number of consumers, downstream criticality and change frequency, to decide on the good datasets to specify in the contract. A third line is the automated contract inference based on datasets and pipeline code to incur less authoring, and automated consumer discovery based on lineage analysis such that the blast radius of a proposed change can be reported in-line in the producer's pull request. Addition of streaming interfaces, where schema and semantics are continuously evolving not at discrete release points, is open, and integration between contract state and catalogue and observability tooling such that contract compliance is an explicit signal in platform monitoring is open as well.

REFERENCES

1. Dehghani, Z., 2022. *Data Mesh: Delivering Data-Driven Value at Scale*. Sebastopol, CA: O'Reilly Media. ISBN 978-1-4920-9239-1.
2. Goedegebuure, A., Kumara, I., Driessen, S., Di Nucci, D., Monsieur, G., van den Heuvel, W.J. and Tamburri, D.A., 2024. Data mesh: a systematic gray literature review. *ACM Computing Surveys*. DOI: 10.1145/3687301. Preprint arXiv:2304.01062.
3. Preston-Werner, T., *Semantic Versioning 2.0.0*. Available at: <https://semver.org/>
4. Qiu, D., Li, B. and Su, Z., 2013. An empirical analysis of the co-evolution of schema and code in database applications. In: *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013)*, pp. 125–135.
5. Vassiliadis, P., 2016. Schema evolution for databases and data warehouses. In: *Business Intelligence (eBISS 2015)*, Lecture Notes in Business Information Processing, vol. 253. Cham: Springer, pp. 1–31.
6. Curino, C.A., Moon, H.J., Tanca, L. and Zaniolo, C., 2008. Schema evolution in Wikipedia: toward a web information system benchmark. In: *Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2008)*.
7. Polyzotis, N., Zinkevich, M., Roy, S., Breck, E. and Whang, S., 2019. Data validation for machine learning. *Proceedings of Machine Learning and Systems (MLSys)*, 1, pp. 334–347.
8. Schelter, S., Lange, D., Schmidt, P., Celikel, M., Biessmann, F. and Grafberger, A., 2018. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment*, 11(12), pp. 1781–1794.
9. Wider, A., Verma, S. and Akhtar, A., 2023. Decentralized data governance as part of a data mesh platform: concepts and approaches. Preprint arXiv:2307.02357.
10. Bitol (Linux Foundation AI & Data), 2025. *Open Data Contract Standard (ODCS)*, version 3.1.0. Available at: <https://bitol-io.github.io/open-data-contract-standard/v3.1.0/>
11. PayPal, 2023. *data-contract-template*. GitHub repository, Apache License 2.0. Available at: <https://github.com/paypal/data-contract-template>
12. Perrin, J.G., 2023. PayPal open sources its data contract template. 1 May 2023. Available at: <https://jgp.ai/2023/05/01/paypal-open-sources-its-data-contract-template/>
13. Bitol, 2023. *Bitol project*, Linux Foundation AI & Data Foundation. Available at: <https://bitol.io/>
14. Bitol, 2025. Bitol announces ODCS v3.1.0: stronger, smarter, and stricter. December 2025. Available at: <https://bitol.io/bitol-announces-odcs-v3-1-0-stronger-smarter-and-stricter/>
15. Wang, R.Y. and Strong, D.M., 1996. Beyond accuracy: what data quality means to data consumers. *Journal of Management Information Systems*, 12(4), pp. 5–33.
16. DAMA International, *DAMA-DMBOK: Data Management Body of Knowledge*, 2nd revised ed. Basking Ridge, NJ: Technics Publications. Available at: <https://www.damadbok.org/>

17. Dong, S., Sahri, S. and Palpanas, T., 2024. Data quality awareness: a journey from traditional data management to data science systems. Preprint arXiv:2411.03007.
18. Shankar, S., Garcia, R., Hellerstein, J.M. and Parameswaran, A.G., 2024. “We have no idea how models will behave in production until production”: how engineers operationalize machine learning. *Proceedings of the ACM on Human-Computer Interaction*, 8(CSCW1), Article 206. DOI: 10.1145/3653697.
19. Apache Software Foundation, *Apache Airflow documentation*. Available at: <https://airflow.apache.org/docs/>
20. Google LLC, *Dataproc documentation*. Google Cloud. Available at: <https://cloud.google.com/dataproc/docs>
21. Apache Software Foundation, *PySpark API documentation*. Available at: <https://spark.apache.org/docs/latest/api/python/>
22. Google LLC, *BigQuery documentation*. Google Cloud. Available at: <https://cloud.google.com/bigquery/docs>
23. GitHub, *GitHub Actions documentation*. Available at: <https://docs.github.com/en/actions>
24. International Organization for Standardization, *ISO 4217 — Currency codes*. Geneva: ISO. Available at: <https://www.iso.org/iso-4217-currency-codes.html>
25. dbt Labs, *Data tests*. dbt documentation. Available at: <https://docs.getdbt.com/docs/build/data-tests>
26. Data Contract CLI, *Enforce data contracts*. Available at: <https://cli.datacontract.com/> — source: <https://github.com/datacontract/datacontract-cli>
27. Soda, *Data contracts*. Soda documentation. Available at: <https://docs.soda.io/soda/data-contracts.html>
28. Great Expectations, *Official documentation*. Available at: <https://docs.greatexpectations.io/docs/home/>
29. Prodduturi, S.M.K., 2023. Leveraging Big Data And Business Intelligence To Revolutionise Corporate Strategy. *International Journal for Research Trends and Innovation*, 8(7).
30. Manoharan, D., 2025. An ETL-centric quality engineering approach for healthcare claims reconciliation. *International Journal of Humanities Science Innovations and Management Studies*, 2(3), pp.32-43.