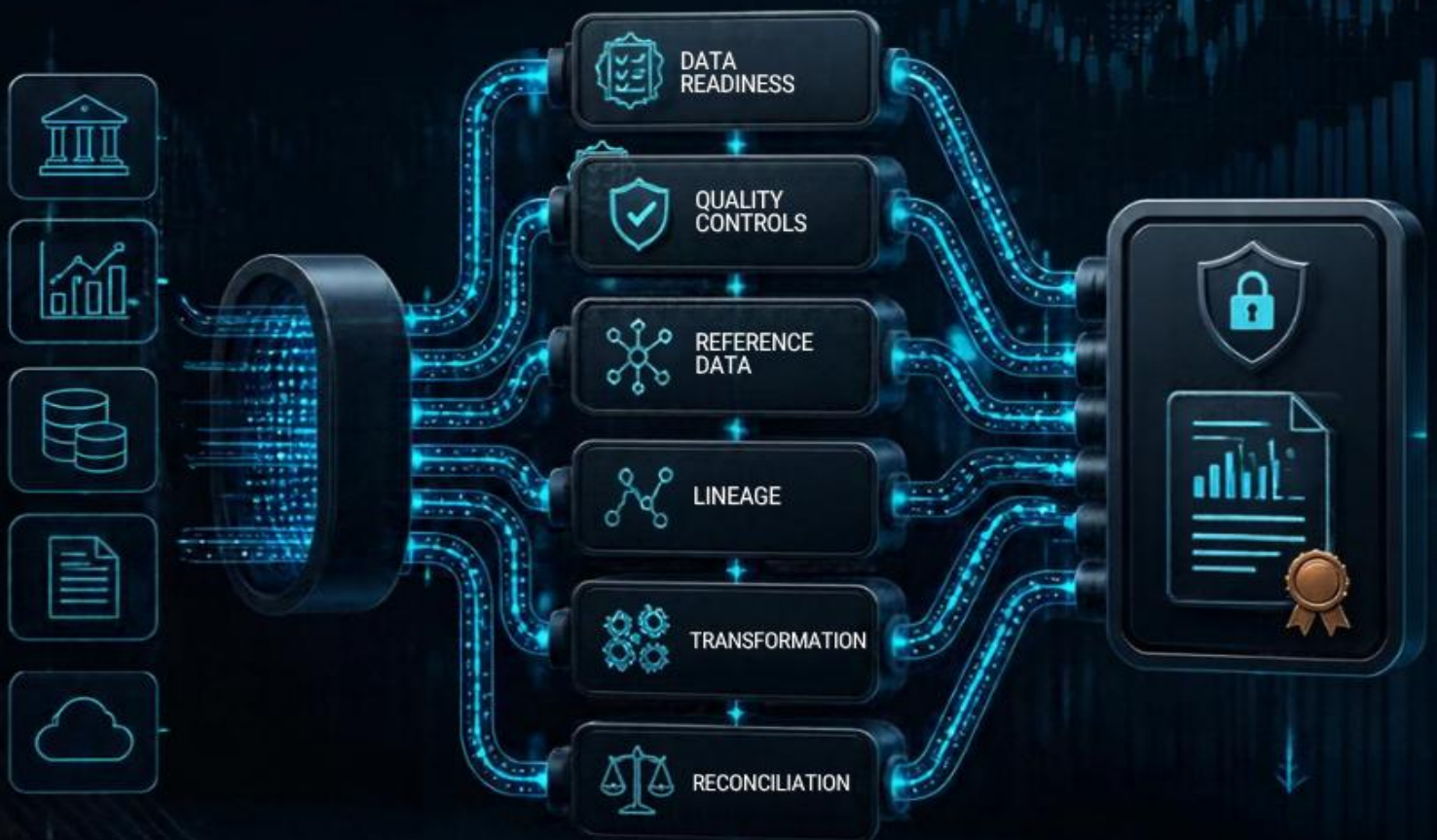


Advanced Data Engineering for Risk and Regulatory Information Systems:

A Comprehensive Study of
Data Readiness, Quality Controls,
Reference Data, Lineage,
Transformation, Reconciliation, and
Regulatory Reporting Architectures



Sudha Donepudi

INDEPENDENT RESEARCHER, USA

*Advanced Data Engineering for Risk
and Regulatory Information Systems: A
Comprehensive Study of Data
Readiness, Quality Controls, Reference
Data, Lineage, Transformation,
Reconciliation, and Regulatory
Reporting Architectures*

Sudha Donepudi

Independent Researcher, USA

Published by
ScienceTech Xplore



Advanced Data Engineering for Risk and Regulatory Information Systems: A Comprehensive Study of Data Readiness, Quality Controls, Reference Data, Lineage, Transformation, Reconciliation, and Regulatory Reporting Architectures

Copyright © 2026 Sudha Donepudi

All rights reserved.

First Published 2026 by ScienceTech Xplore

ISBN 978-93-49929-18-0

ScienceTech Xplore

www.sciencetechxplore.org

The right of Sudha Donepudi to be identified as the author of this work has been asserted in accordance with the Copyright, Designs, and Patents Act of 1988. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise) without the prior written permission of the publisher.

This publication is designed to provide accurate and authoritative information. It is sold under the express understanding that any decisions or actions you take as a result of reading this book must be based on your judgment and will be at your sole risk. The author will not be held responsible for the consequences of any actions and/or decisions taken as a result of any information given or recommendations made.



978-93-49929-18-0

Printed and Bounded by
ScienceTech Xplore, India

About the Author



Sudha Donepudi, Senior Data Analyst with 20 years of IT experience specializing in end-to-end data validation, requirements gathering, planning, and execution across the Financial (BFS), Telecommunications, and Healthcare industries. Proven track record of managing massive data volumes by bridging legacy data warehouses (Oracle, Teradata, DB2, Sybase) with modern Big Data environments (Hadoop, Hive, Impala). Recognized expert in FED regulatory reporting frameworks, retail risk data profiling, and account-level data quality governance. Proficient at translating complex Business Requirement Documents (BRDs) into highly optimized ETL mapping logic, maintaining data lineage, and establishing automated validation strategies using tools like QTP and QlikView.

With over 10 years of experience in Banking and Financial Services (BFS), Sudha Donepudi specializes in data analysis, data validation, and financial data architecture across hybrid environments, including Hadoop, Oracle, and Teradata. Her expertise includes gathering business requirements through Business Requirement Documents (BRDs), implementing data readiness processes, ensuring data quality, validating Federal regulatory reporting data, and maintaining end-to-end data lineage. She has extensive experience in ETL processes using Informatica and Ab Initio, SQL query optimization, UNIX and Shell scripting, and database management across Oracle, DB2, Sybase, and Teradata. Her technical proficiency also encompasses distributed file systems, Hadoop cluster data transfer and replication, high-volume data processing, and data integration to establish a Single Source of Truth (SSOT). Additionally, she is skilled in QlikView reporting, QTP automation, Hive and Impala queries, JIRA-based issue tracking, performance optimization, and enforcing enterprise architecture standards while providing technical guidance to junior engineers and stakeholders.

Preface

In an era where organizations increasingly depend on data-driven decision-making, the ability to manage, validate, transform, and govern information reliably has become essential to effective risk management and regulatory compliance. Financial institutions and other regulated enterprises operate within complex data ecosystems that require accuracy, consistency, transparency, and traceability across multiple systems. As data volumes grow and architectures evolve, robust data engineering practices are fundamental to ensuring that information remains trustworthy, accessible, and fit for regulatory and business purposes.

Advanced Data Engineering for Risk and Regulatory Information Systems: A Comprehensive Study of Data Readiness, Quality Controls, Reference Data, Lineage, Transformation, Reconciliation, and Regulatory Reporting Architectures explores the principles, methodologies, and architectural practices that support reliable data management in risk and regulatory environments. The book focuses on the interconnected processes of data readiness, data quality assurance, reference data validation, end-to-end lineage, transformation logic, reconciliation, and regulatory reporting. It emphasizes the importance of establishing dependable data flows, maintaining consistency across heterogeneous platforms, and implementing systematic controls that strengthen data integrity throughout the information lifecycle.

Particular attention is given to the relationship between traditional relational database management systems and modern data engineering environments, including distributed processing platforms and enterprise data warehouses. The discussion highlights the role of structured validation frameworks, metadata management, traceability mechanisms, and standardized transformation processes in addressing data discrepancies and supporting accurate downstream reporting. By examining these interconnected components, the book presents a comprehensive perspective on how data engineering contributes to operational reliability, regulatory accountability, and informed risk assessment. A central theme of this work is that regulatory reporting excellence begins long before a report is generated. It depends on the quality of source information, the effectiveness of data integration, the transparency of transformation rules, and the strength of reconciliation and validation controls. Establishing a reliable data foundation therefore requires coordinated processes, well-defined architectural standards, and continuous monitoring across the entire data lifecycle.

This book is intended for data engineers, data analysts, data architects, quality assurance professionals, risk management specialists, regulatory reporting teams, technology consultants, and researchers interested in enterprise data systems. It may also serve as a practical reference for professionals seeking to understand the relationship between data engineering principles and the demands of financial risk and regulatory information management. Through its integrated treatment of data architecture, quality controls, transformation, lineage, reconciliation, and reporting, this work aims to encourage a disciplined and systematic approach to enterprise data engineering. The broader objective is to demonstrate how well-designed data processes can improve trust in information, strengthen governance, and support the development of resilient and transparent regulatory information systems.

As organizations continue to modernize their data infrastructures, the principles discussed in this book remain relevant to building systems that are scalable, auditable, and responsive to evolving business and regulatory requirements. It is hoped that this work will contribute to professional understanding, encourage further exploration, and provide readers with valuable perspectives for advancing data engineering practices in complex enterprise environments.

Acknowledgement

I would like to express my sincere gratitude to everyone whose encouragement, support, and professional insights have contributed to the development of this book, *Advanced Data Engineering for Risk and Regulatory Information Systems: A Comprehensive Study of Data Readiness, Quality Controls, Reference Data, Lineage, Transformation, Reconciliation, and Regulatory Reporting Architectures*.

My professional journey across data engineering, data analytics, quality assurance, and enterprise information systems has provided valuable opportunities to understand the importance of data accuracy, integrity, and governance in complex business environments. The knowledge and experience gained through working with diverse technologies, teams, and enterprise systems have helped shape the perspectives presented in this work.

I am grateful to the colleagues, technical professionals, business analysts, data specialists, and quality assurance teams with whom I have had the opportunity to collaborate throughout my career. Their shared experiences, problem-solving approaches, and commitment to maintaining reliable information systems have contributed to my understanding of the practical challenges involved in data readiness, validation, transformation, reconciliation, and regulatory reporting.

I also extend my appreciation to the broader professional and technical community whose continuing contributions to data engineering, database technologies, distributed processing, and information governance have helped advance the field. The collective efforts of practitioners and researchers provide an important foundation for developing more reliable, transparent, and efficient enterprise data architectures.

My heartfelt thanks go to my family and well-wishers for their encouragement, patience, and support throughout the process of developing this book. Their understanding and motivation have been invaluable in sustaining my commitment to this work.

Finally, I would like to thank all the readers, practitioners, and researchers who continue to explore ways to improve data quality, strengthen regulatory information systems, and advance responsible data management practices. I hope this book offers useful insights, encourages professional discussion, and serves as a meaningful reference for those working to build trustworthy and resilient data environments.

Sudha Donepudi

Independent Researcher, USA

Table of Contents

About the Author	i
Preface	ii
Acknowledgement	iii
PART 1	
The Evolution and Foundations of Regulatory Data Engineering	1
Enterprise Risk Information Systems and Regulatory Data Architecture	11
Data Readiness Assessment, Profiling, and Source-System Discovery	22
PART 2	
Data Quality Dimensions, Validation Frameworks, and Control Engineering	31
Reference Data Management, Master Data, and Classification Standards	41
Metadata Management, Data Lineage, and Information Traceability	50
PART 3	
Data Ingestion, Integration, and Regulatory Data Pipelines	57
Data Transformation, Standardization, and Business Rule Execution	66
Data Reconciliation, Completeness, and Financial Control Frameworks	74
Exception Management, Data Remediation, and Audit Evidence	82
PART 4	
Regulatory Reporting Data Models and Information Delivery	88
Regulatory Reporting Pipelines, Submission Controls, and Validation	95
Data Governance, Security, Privacy, and Regulatory Accountability	102
Scalable, Resilient, and Observable Regulatory Data Platforms	109
PART 5	
Enterprise Implementation Methodologies, Testing, and Operational Readiness	115
Architectural Trade-Offs, Integrated Case Studies, and Future Directions	121
Appendix A	128
Appendix B	131
Bibliography	135

CHAPTER 1

The Evolution and Foundations of Regulatory Data Engineering

Regulatory information systems occupy an unusual position in enterprise engineering. They are the point at which the operational reality of a financial institution is translated into a form that supervisors, auditors and markets will accept as evidence. This chapter establishes the vocabulary, the historical trajectory and the architectural premises that the remainder of the book develops in detail. It argues that regulatory data engineering is not a reporting specialism bolted onto a general data platform, but a discipline with its own control logic, its own notion of correctness and its own evidentiary obligations.

Regulatory data engineering is the discipline of designing, building, operating and evidencing the data pipelines, data models and control frameworks through which an institution demonstrates its financial and risk condition to an external authority. The phrase is deliberately composite. It is data engineering because the primary artefacts are datasets, schemas and transformations. It is regulatory because the acceptability of those artefacts is judged against defined disclosure obligations rather than against internal analytical preference. And it is engineering rather than administration because the obligations are long-lived, the volumes are large, the deadlines are immovable and the consequences of silent failure are severe.

The distinction between management reporting and regulatory reporting is worth stating precisely at the outset, because much of the difficulty encountered by institutions arises from treating them as the same problem. Management reporting is designed to inform a decision. Its consumer is internal, its granularity is chosen for analytical usefulness, its definitions evolve as the business evolves, and a degree of imprecision is tolerable because the decision-maker can interrogate the number and apply judgement. Regulatory reporting is designed to discharge an obligation. Its consumer is external, its granularity is prescribed, its definitions are fixed by the reporting framework, and imprecision is not tolerable because the number will be compared against prior periods, against peers and against other disclosures by the same institution.

The Historical Trajectory of Regulatory Reporting Systems

To understand why contemporary regulatory data architectures, look the way they do, it is necessary to understand the sequence of problems that produced them. That sequence is not a story of steady improvement. It is a story of successive architectural bandages applied to a problem that was misdiagnosed for nearly three decades.

During the 1980s, reporting was performed by extracting fixed-format files from mainframe ledgers and loading them into purpose-built reporting programs. The programs encoded the reporting logic directly, often in procedural code tied to a specific report layout. This approach had the merit of transparency: a competent analyst could read the code and determine exactly how a reported figure had been produced. It had the defect of extreme rigidity. A change to the reporting framework required a change to program code, a regression test cycle and a release. The mapping between source and report existed only in the code and in the memory of the people who had written it.

The 1990s introduced the enterprise data warehouse as the organising concept for corporate information. The warehouse promised a single, integrated, subject-oriented store from which multiple reporting requirements could be satisfied. In practice, the promise was partly kept. Warehouses did consolidate source data and did decouple reporting logic from source transactions. But the consolidation was typically achieved by modelling the business rather than the reporting obligation, and the resulting structures were optimised for the questions the business asked rather than the disclosures the regulator demanded.

The 2000s added tool-based extract-transform-load (ETL) processing and the emergence of master data management. ETL tooling externalised transformation logic from program code into metadata, which was a genuine advance for maintainability. Conceptual modelling approaches for ETL processes formalised the mapping between sources, transformations and targets, giving engineers a vocabulary for design that had previously been implicit [19]. Master data management addressed the chronic problem of inconsistent entity identity across systems, which is discussed at length in Chapter 5. Yet the regulatory reporting obligations of this period were still, with important exceptions, comparatively modest in granularity and volume.

The decisive change came after the 2007 to 2009 financial crisis. Supervisors had discovered during the crisis that they could not obtain reliable, timely information about the risk profile of the largest institutions. Aggregation was slow, definitions were inconsistent, and the provenance of figures reported to the supervisor could not be established. The Basel Committee on Banking Supervision responded with a set of principles for effective risk data aggregation and risk reporting, commonly known by their reference number, BCBS 239 [49]. The Financial Stability Board conducted a thematic review of implementation and found progress to be slower and shallower than institutions had claimed [52].

BCBS 239 is worth reading carefully because it is not, in essence, a technology standard. It is a governance and capability standard that happens to have profound technological consequences. Its principles address governance and infrastructure, data architecture, accuracy and integrity, completeness, timeliness, and the adaptability of aggregation capability to stress and change. The relevance of the principles to this book is that each of them implies an architectural capability, and each capability has an engineering cost. Figure Risk Data Aggregation Principles Mapped to Architecture Capabilities sets out the mapping that this book uses as its analytical spine.

The Convergence of Reporting Obligations

A second historical force has shaped the contemporary landscape: the proliferation and convergence of reporting obligations. A large institution in 2020 does not produce one regulatory return. It produces prudential returns under Basel III [50] and, in Europe, under the capital requirements regulation. It produces financial statements under IFRS, including fair value and expected credit loss disclosures under IFRS 9 [58]. In Europe it produces insurance returns under Solvency II where it carries insurance business [56], and it produces statistical and payments data under SDMX and ISO 20022 frameworks [48], [44]. It responds to ad hoc supervisory data requests that may arrive with days of notice.

These obligations differ in their legal basis, their cadence, their granularity and their technical format. What they share, increasingly, is a common substrate. The same counterparty identity, the same instrument classification, the same exposure measurement and the same legal entity hierarchy underpin all of them. The architectural implication is significant. An institution that builds a separate pipeline for each obligation multiplies its control surface, its reconciliation burden and its audit exposure. An institution that builds a common substrate and then projects it into each obligation's presentation layer pays a design cost upfront and gains a durable reduction in cost of change.

Regulatory Data Lifecycle traces this evolution across four decades. The important observation is not that each stage was superseded. It is that each stage left a residue. Mainframe extracts still exist in many institutions, wrapped in modern orchestration. Warehouse models coexist with data lakes. Tool-based ETL coexists with code-based pipelines. The engineering challenge is not to choose a single paradigm but to impose coherent control over the mixture that history has produced.

From Reporting Output to Reporting Evidence

The final conceptual shift is the most consequential for engineering practice. Under the older model, the deliverable of the reporting function was a submitted figure. Under the contemporary model, elaborated by the principles in BCBS 239 and by supervisory practice after them, the deliverable is a figure plus the evidence that the figure is correct. This shift changes what must be built. It is no longer sufficient to compute a number and transmit it. The institution must be able to demonstrate, on demand and after the fact, which source records contributed to the number, which transformation rules were applied, which reference data versions were in force, which validation checks were executed and with what result, who approved the release, and whether the number would be reproduced identically if the pipeline were re-run. Each of

these requirements corresponds to a system capability: column-level lineage capture, rule versioning, reference data versioning, control execution logging, workflow-based approval and immutable archival with reproducible execution.

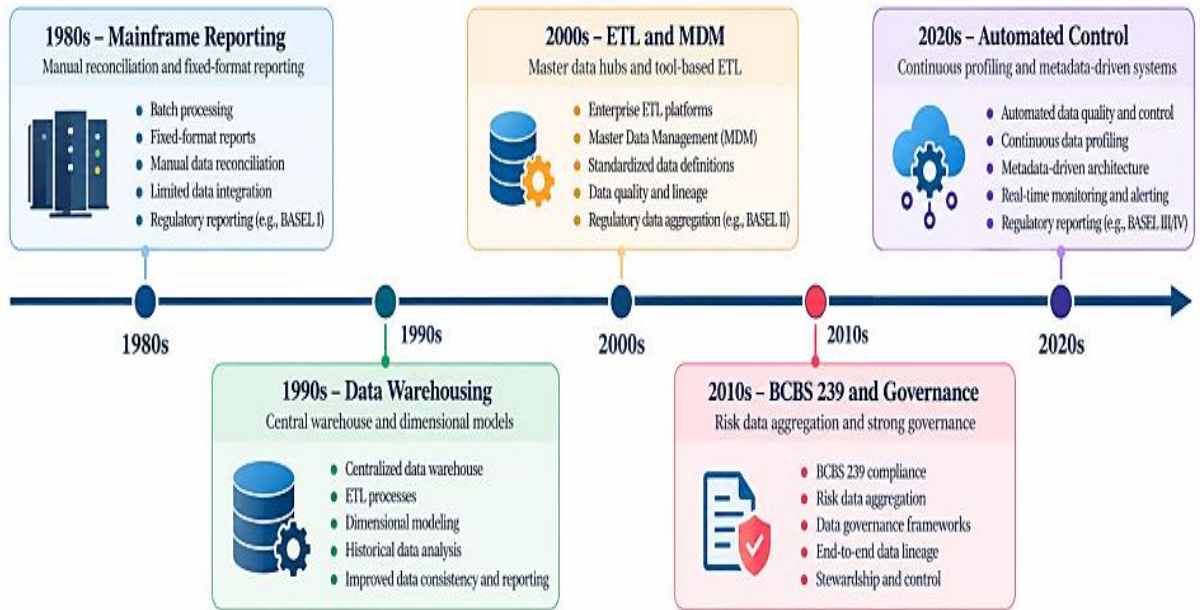


Figure 1.1 Evolution of Regulatory Data Engineering

The notion of provenance developed in the database research literature provides the theoretical vocabulary for part of this requirement. The distinction between why-provenance, how-provenance and where-provenance, characterised by Buneman, Khanna and Tan and surveyed comprehensively by Cheney, Chiticariu and Tan, maps closely onto the questions that supervisors and auditors ask of a reported figure [27], [28]. Why-provenance answers which source rows contributed. Where-provenance answers which source column a value was copied from. How-provenance answers the derivation sequence. Figure Column-Level Lineage Graph presents this mapping explicitly, and Chapter 6 develops it into an architectural specification.

Four Foundational Premises

The argument of this book rests on four premises. They are stated here so that the reader can test them against the material that follows.

The first premise is that correctness in regulatory data is defined relationally rather than absolutely. There is no property of a single record that makes it correct in isolation. A record is correct if it is consistent with the rules of the reporting framework, consistent with the reference data in force, consistent with other records describing the same underlying event, and consistent with the control totals that the institution's financial systems assert. Regulatory data quality is therefore a property of a set of relationships, and the engineering discipline that produces it is the discipline of establishing, monitoring and repairing those relationships.

The second premise is that control must be continuous rather than terminal. A validation performed once at the end of a pipeline establishes a fact about a single execution. It provides no assurance about the next execution. The architectural response is to embed validation as an executable, versioned, continuously monitored artefact that produces a result on every run and whose history is retained. This is the subject of Chapters 4 and 12.

The third premise is that traceability must be designed rather than reconstructed. It is possible, in principle, to reconstruct the lineage of a figure after the fact by reading code and interrogating engineers. In practice this consumes days of skilled effort, produces an answer of uncertain completeness, and cannot be performed for historical periods once the code has

changed. An institution that requires traceability as a routine capability must capture it at execution time as a first-class output of the pipeline. Chapter 6 develops the mechanics.

The fourth premise is that the cost of change dominates the cost of construction. A regulatory data platform is built once and modified continuously. Reference data changes, reporting frameworks change, source systems are replaced, obligations are added. The architecture that minimizes the cost of the first delivery is very often the architecture that maximises the cost of the two hundredth change. Figure Architectural Trade-Offs in Regulatory Data Platforms positions the common architectural choices against exactly this trade-off.

The Regulatory Data Lifecycle

It is useful to have a single organizing model for the material in this book. Enterprise Regulatory Data Architecture presents the regulatory data lifecycle as a closed loop of six stages: discovery and profiling; ingestion and landing; standardization and enrichment; validation and control; reconciliation and reporting; and retention and evidence.

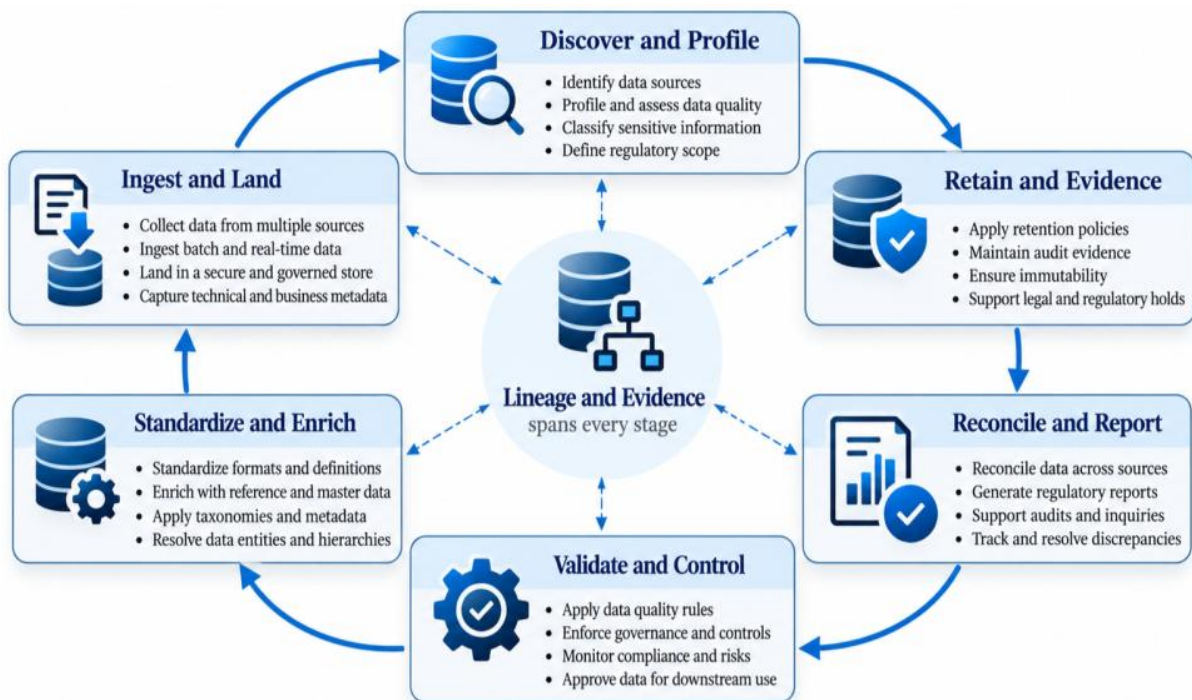


Figure 1.2 The Regulatory Data Lifecycle

The lifecycle is a loop rather than a line because each stage generates information that feeds back into earlier stages. Profiling results inform the design of validation rules. Validation failures reveal defects in source systems that must be remediated at origin. Reconciliation variances reveal errors in transformation logic that must be corrected and re-deployed. Stewardship decisions taken during exception resolution change reference data, which alters the behaviour of every downstream stage. An architecture that does not support this feedback is an architecture that will accumulate unresolved defects.

Each stage of the lifecycle corresponds to a distinct engineering concern, and each has a natural failure mode. Discovery fails by omission: a source system that contributes to the reported figure is never identified, and its data is silently absent. Ingestion fails by partiality: a feed is truncated, a file arrives late, a schema changes without notice. Standardisation fails by inconsistency: the same concept is represented differently in two pipelines. Validation fails by staleness: rules describe a reporting framework version that has been superseded. Reconciliation fails by unexamined tolerance: variances are within threshold but wrong in a consistent, offsetting direction. Retention fails by irreproducibility: the data is retained but the code and reference data required to reproduce the calculation are not.

The lifecycle model also clarifies the boundary of this book. It is concerned with the data and control architecture that supports reporting. It is not concerned with the mathematical specification of risk measures, which belongs to the risk modelling literature, nor with the design of the reporting templates themselves, which is prescribed by the relevant authority. What it is concerned with is the machinery that moves data from source to submission with controlled, evidenced and reproducible behaviour.

Sources of Complexity

Four characteristics of the regulatory reporting problem drive most of the architectural complexity encountered in practice.

The first is multiplicity of source systems. A single reported figure, such as the total exposure to a counterparty group, may draw on the general ledger, a lending system, a trading platform, a collateral management system and a credit risk engine. Each has its own identifier scheme, its own treatment of dates and its own notion of what constitutes a complete record. Integrating these sources into a coherent view requires schema matching, which has been studied extensively in the data integration literature [16], [17], and identity resolution, which is treated in the entity resolution literature [12], [13].

The second is asymmetry between source and target semantics. Source systems are designed to support a process, not to disclose a position. A lending system knows what an account officer did on a given day. It does not necessarily know how the resulting exposure should be classified under a regulatory taxonomy, which legal entity should bear it. The transformation layer must supply this semantic bridging, and the rules that perform the bridging are themselves governed artefacts requiring version control and impact assessment.

The third is the temporal dimension. Regulatory reporting is fundamentally about dates. A return for a period reflects positions as at a date, movements over a period and classifications in force at that time. Source systems overwrite history. Reference data is updated in place. This creates a persistent tension between the operational present and the reporting past. The architectural responses, which include bitemporal modelling, effective-dated reference data and immutable landing zones, are developed in Chapters 5, 7 and 9.

The fourth is the audit relationship. An institution's regulatory reporting process is examined by internal audit, by external auditors in connection with financial statement assertions, and by supervisors. Each examines a different aspect of the process, but all require evidence. The evidence requirement is not uniform. A supervisor may ask a specific question about a specific figure for a specific past period. The architecture must be able to answer such a question without a project.

The Regulatory Data Architecture Blueprint

The remainder of this book is organised around a reference design, which is referred to throughout as the Regulatory Data Architecture Blueprint. The Blueprint is a layered architecture whose layers correspond to the lifecycle stages described above, with governance and traceability treated as cross-cutting concerns that appear at every layer rather than as a separate tier. Figure Evolution of Regulatory Data Engineering presents the Blueprint at its highest level of abstraction.

The Blueprint has seven layers. The source systems layer contains the operational and analytical systems of record and the external data services that supply market and reference information. The ingestion and integration layer provides the mechanisms by which data leaves those systems and enters the platform, including batch extraction, change data capture, interface-based retrieval and file landing. The quality and control layer contains the executable validation artefacts, the profiling capabilities that inform them, the exception queue that receives their failures and the reporting that makes their results visible. The reference and master data layer holds the governed classifications, hierarchies and entity identities that give meaning to transactional values. The aggregation and storage layer contains the physical stores in which data is held for reporting, including the enterprise warehouse, risk data marts and immutable archives. The reporting and submission layer contains the calculation, rendering and transmission capabilities. The governance and assurance layer contains the policy, lineage, access control and audit capabilities that make the whole stack demonstrable.



Figure 1.3 Enterprise Regulatory Data Architecture

Two properties of the Blueprint deserve emphasis. The first is that it is a logical rather than a physical design. The layers describe responsibilities, not deployment units. A large institution may implement several layers in a single technology platform; a smaller institution may implement a single layer across several platforms. What matters is that each responsibility is explicitly assigned and that the interfaces between responsibilities are defined.

The second is that the Blueprint is deliberately conservative with respect to technology fashion. Its layers are recognisable in architectures built on relational warehouses, on distributed processing frameworks and on contemporary coupling of object storage with batch and streaming engines [22], [24]. The technology choices within a layer are separable from the architecture, and Chapter 14 treats the principal choices and their trade-offs explicitly.

Reading the Blueprint as a Set of Obligations

The Blueprint is more useful when read as a set of obligations than as a diagram. Each layer obliges the institution to make a decision and to be able to evidence it. The source systems layer obliges the institution to be able to enumerate the systems that contribute to each reported figure, to identify the responsible owner of each, and to state the extraction mechanism and its timing. An institution that cannot answer the question which systems contribute to this figure? cannot satisfy the traceability obligations discussed in Chapter 6. The ingestion layer obliges the institution to define what constitutes a complete delivery from each source, to detect the failure modes that violate completeness, and to define the recovery path. Figure Ingestion Failure Modes and Controls sets out the recognised failure modes and the corresponding controls.

The control layer obliges the institution to express its expectations about data as executable rules, to define thresholds that distinguish acceptable from unacceptable results, and to operate a process for handling failures. The control layer is where the abstract notion of data quality becomes an operational reality, and Chapter 4 develops the design principles. The reference and master data layer obliges the institution to establish authoritative definitions for the classifications and identities that appear in its reports, and to manage changes to those definitions with formal impact assessment. It is the layer most often under-invested relative to its contribution to reporting defects, a point discussed in Chapter 5. The storage and aggregation layer obliges the institution to demonstrate that the arithmetic relationships asserted in its reports hold in its data, and that the data used to produce a past report can be reproduced. The reporting layer obliges the institution to demonstrate that the transformation from internal representation to external presentation is unambiguous and controlled.

The governance layer obliges the institution to demonstrate that decisions about data are made by identifiable people following documented processes.

Standards and Their Role

Regulatory data engineering is unusual among engineering disciplines in the extent to which its vocabulary is codified in formal standards. Three families are particularly relevant and are used throughout this book. The data quality standards define the characteristics of data that may be assessed and the processes by which assessment is managed. ISO/IEC 25012 provides a general data quality model for data held in structured form, distinguishing inherent, system-dependent and dependent characteristics [39], and ISO/IEC 25024 provides measures for those characteristics [70]. ISO 8000-61 defines a process reference model for data quality management, which is of practical value because it separates the activities that an organisation must perform from the technology it uses to perform them [40]. ISO 8000-110 addresses the exchange of master data characteristics [41]. The standards are not prescriptive about architecture, which is precisely why they are useful as a common vocabulary.

The reference data standards define the coded representations of the concepts most commonly exchanged between institutions and authorities. ISO 17442 defines the legal entity identifier [43], ISO 4217 defines currency codes [69], ISO 3166 defines country codes [68] and ISO 8601 defines date and time representations [67]. Chapter 5 discusses the practical governance of these standards within an institution, including the common failure of partial adoption. The reporting and messaging standards define the technical form in which data is exchanged. XBRL provides a syntax for structured business reporting with an accompanying taxonomy mechanism [47]. SDMX provides a framework for statistical data and metadata exchange [48]. ISO 20022 provides a metamodel for financial message definitions and a growing library of specific message standards [44]. These standards matter architecturally because they impose structure on the presentation layer and because their taxonomy versioning behaviour must be accommodated by the release process described in Chapter 12.

A fourth family, the supervisory publications, is not a standard in the formal sense but functions as one in practice. BCBS 239 defines expectations for risk data aggregation [49]. SR 11-7 defines expectations for model risk management, which is relevant where reporting figures depend on models [53]. The COSO framework defines a general approach to internal control over reporting [54]. Chapter 13 positions these requirements within the governance architecture. The book treats standards as design inputs rather than as objectives in themselves. Conformance to a standard is valuable when it reduces the cost of integration, increases the credibility of evidence or provides a vocabulary that reduces misunderstanding. It is not valuable as an end in itself, and the pursuit of conformance beyond what the reporting obligation requires is a common source of wasted engineering effort.

What This Book Commits To

This book is written from an engineering perspective and takes positions. Stating them in advance allows the reader to evaluate the reasoning rather than infer the position. It holds that data quality is best addressed by prevention at source supplemented by detection downstream, and that investment in detection that would be unnecessary under prevention is a symptom of unresolved architectural debt. It holds that reference data governance is disproportionately valuable relative to its cost, because reference data defects propagate to every downstream consumer while reference data corrections propagate instantly. It holds that lineage should be captured automatically at execution time rather than maintained as documentation, because documentation drifts and execution-time capture does not. It holds that reconciliation is not a compliance overhead but the primary mechanism by which the correctness of an aggregation architecture is established, and that a variance explained is worth more than a variance suppressed. It holds that the cost of change is the dominant cost in regulatory data platforms and that architectural choices should be evaluated primarily against that criterion.

It also holds, and this is important, that not every institution should build the same thing. The Blueprint is presented as a design space with stated trade-offs rather than as a prescription. A small institution reporting a limited set of obligations should not construct the seven-layer architecture described above. Chapter 16 develops the trade-off framework explicitly, and Chapter 15 provides a phased implementation methodology that scales the approach to the obligation at hand. Risk Data Aggregation Target Architecture presents the target architecture in the form used throughout the book, with each

component accompanied by the concerns it addresses. Chapter 2 develops that architecture in full detail, establishing the relationship between supervisory expectations and engineering capabilities that the remainder of the book assumes.

Finally, a note on the treatment of empirical claims. This book contains no original empirical research. Where it cites a study, the citation is to the published work. Where it describes architectural patterns, the description is of designs that are in use across the industry, presented in generalised form and without reference to specific institutions. Where it makes a recommendation, the recommendation is identified as a design proposition rather than as an established requirement. The distinction matters, because the reader must be able to tell which statements are reports of what the rules require and which are engineering judgements about how best to satisfy them.

Worked Example: Tracing a Reported Figure to Its Origin

An illustration makes the Blueprint's obligations concrete. Suppose a supervisor asks how the reported total exposure to a named counterparty group was derived, for a period nine months past. The question appears simple and it exercises nearly every capability described in this book. The institution must first identify the reporting line. The data point is a specific row and column of a specific return version, which identifies the taxonomy release under which it was submitted, and through the mapping specification identifies the source attribute. The mapping specification names the aggregation layer table, which names the derived-layer measure, which names the rule that computed it. The rule carries a version identity, and the rule version in force for the period is retrieved from the rule inventory rather than inferred from the currently deployed code.

The rule's declared inputs identify the standardized-layer attributes it consumed. Each of those attributes traces to one or more source columns through the recorded lineage edges. The counterparty group membership, which determines the population, is resolved from the legal entity hierarchy as it stood at the reporting date, which requires the hierarchy to be effective-dated. The exposure measurement depends on a risk classification that resolves against a reference data set, and the version of that set in force at the reporting date is retrieved from the reference data version record attached to the pipeline execution. Finally, the control results for the period are retrieved: the completeness check against the ledger control total, the reconciliation against the independent source extract, and the disposition of any exceptions raised. The answer assembled is not merely a number but a defensible account of how the number was produced and why the institution believes it is correct. An institution in which each of these retrieval steps is a query answers the question in hours. An institution in which they are investigations, interviews and code reading answers it in weeks, with an answer whose completeness cannot be demonstrated. The difference between the two is architectural and is determined long before the question is asked.

Common Misconceptions About Regulatory Data Engineering

Four misconceptions recur in practice and each leads to a predictable class of failure.

The first is that regulatory reporting is a downstream consumer of data and that improving the reporting pipeline is the way to improve reporting quality. In reality a substantial proportion of reporting defects originate in source systems, in reference data and in master data, and a reporting function that can only act downstream will spend its effort compensating for defects it cannot prevent. The architectural response is to give the reporting function a governed requirement on source owners, supported by the completeness and semantic controls that make a source-side defect visible where it originates.

The second is that data quality is achieved by purchasing a tool. Tooling is necessary and it is not sufficient. A validation tool with no rule inventory, no owned thresholds and no exception process produces reports that no one acts upon. The investment that produces quality is in the definition of expectations, the assignment of ownership and the operation of a remediation process; the tool executes what those produce.

The third is that lineage can be documented. Documentation describes what the system did when the documentation was written. Lineage as a capability describes what the system did when it executed, and it is only obtainable by capturing it at execution or by deriving it from a declarative specification of the transformation. An institution that plans to document its lineage will find the documentation obsolete before it is complete.

The fourth is that reconciliation and validation are alternative controls and that a strong validation framework makes reconciliation unnecessary. They detect different classes of error. Validation detects violations of expectations the institution has articulated. Reconciliation detects the consequence of expectations it has not articulated, which is precisely the class of error that produces a supervisory question.

The Economic Case for Regulatory Data Engineering

An argument is sometimes made that regulatory data engineering is a cost of compliance with no return beyond the avoidance of supervisory sanction. The argument is incomplete, and stating why clarifies the incentives that shape the architecture.

The first return is the reduction of restatement cost. An institution that discovers a defect after submission must correct the figure, notify where required, and explain the correction. The cost of a restatement is dominated not by the recomputation but by the investigation required to establish what went wrong and the evidential burden of demonstrating that the corrected figure is right. Both costs fall sharply where lineage, rule versioning and retained evidence exist as capabilities.

The second return is the reduction of the manual effort consumed by the reporting cycle. Where reconciliation is manual, the effort grows with the number of obligations and the frequency of change. Where it is designed and automated, the marginal cost of an additional obligation is a configuration rather than an exercise.

The third return is the improvement in management information. The governed aggregation layer that produces the reported figures also produces the figures the institution uses to run itself, and the two are reconciled to each other. Where they are separate, management acts on figures that the supervisor would not accept, and the inconsistency is discovered externally.

The fourth is resilience to examination. The cost of an examination is a function of the effort required to answer questions. An institution that can answer them by query spends the examination demonstrating its controls; an institution that must answer them by investigation spends it demonstrating the limits of its knowledge.

The Failure Mode of Programme-Led Improvement

Improvement programmes frequently produce the appearance of progress and no durable capability, and the mechanism is consistent enough to describe.

The programme establishes a target state, secures funding, assembles a team and delivers a set of artefacts: a strategy, a target architecture, a governance framework, a tool selection and a remediation plan. The artefacts are frequently of good quality. What does not survive is the connection between them and the operating routine, because the programme's members return to their functions and the routine reverts to what it was.

The design that avoids this connects every deliverable to a routine that will exist after the programme. A rule inventory is connected to the change process for transformation code. A lineage capability is connected to the impact assessment required before a source change. A control framework is connected to the period-end closure. An exception process is connected to the reporting calendar. The test of a programme's output is not whether the artefacts exist but whether a specific routine operates differently because of them, and that test can be applied to each deliverable before the programme closes.

Figure 1.4 illustrates an end-to-end enterprise architecture for transforming data from core banking systems, risk systems, and external data providers into validated risk information and regulatory reporting outputs. The architecture is organized into four major layers: enterprise data sources, data engineering, control and processing, and regulatory delivery.

The Data Engineering Layer performs data ingestion, standardization, and integration. Raw records are collected from multiple sources, cleaned and validated, transformed into consistent formats, and integrated into a unified dataset. These

activities help establish a consistent data foundation for downstream risk calculations and regulatory reporting while supporting the harmonization of heterogeneous source information.

The Control and Processing Layer applies data-quality profiling, validation rules, and risk data processing to identify inconsistencies and ensure that records satisfy defined business and policy requirements. The Regulatory Delivery Layer then uses the resulting risk information to generate regulatory reports, dashboards, and filings. An audit repository preserves reporting evidence and audit trails, supporting traceability, review, and reconstruction of reporting activities. The architecture emphasizes a controlled flow of information from source systems to reporting outputs, with quality checks and evidence retention supporting regulatory accountability.

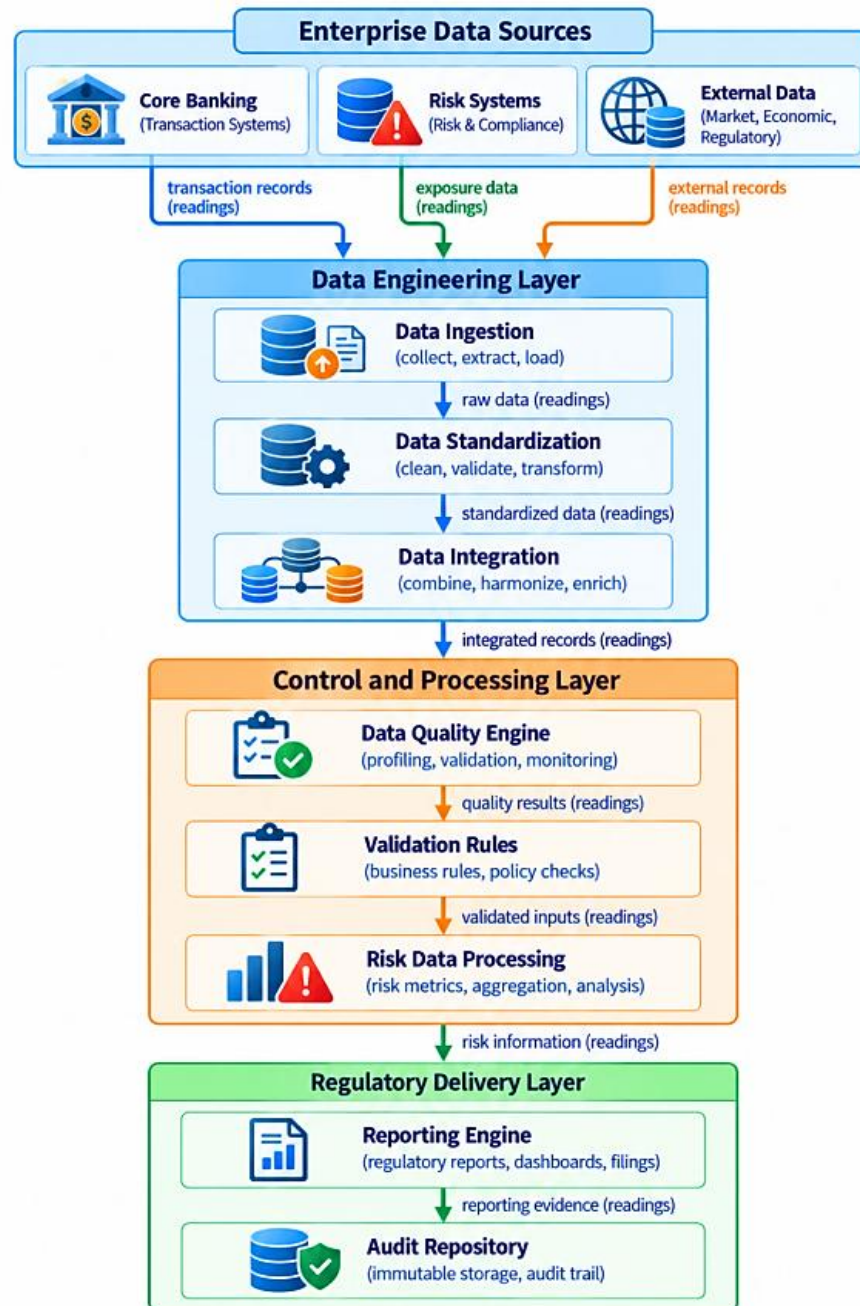


Figure 1.4: Enterprise Regulatory Data Engineering, Control, and Reporting Architecture

CHAPTER 2

Enterprise Risk Information Systems and Regulatory Data Architecture

The previous chapter established why regulatory data engineering deserves treatment as a discipline in its own right. This chapter converts that argument into architecture. It describes the structure of enterprise risk information systems, sets out the reference design that the book calls the Regulatory Data Architecture Blueprint, and connects each architectural component to the supervisory expectations that make it necessary. The objective is not to present a single correct design but to establish the design space, the responsibilities that must be assigned within it, and the criteria by which a design may be judged.

An enterprise risk information system is the collection of data, processes, models and controls through which an institution measures, aggregates and reports its risk profile. The definition is deliberately broad because the system is rarely a single platform. In most institutions it is a federation of source systems, a set of integration and transformation pipelines, one or more aggregation engines, a reporting layer and a governance apparatus that spans all of them. The architectural question is not whether this federation exists but whether its behaviour is coherent, observable and evidential.

The distinction between an information system and a data platform is worth drawing. A data platform is infrastructure: storage, compute, orchestration and access management. An information system is a platform plus the semantics, rules and controls that make its contents mean something specific. A regulatory data platform that stores correct values but cannot explain what they represent, how they were derived and whether they are complete is not an information system in this sense. Much of the difficulty encountered in practice stems from investments that build platforms and assume that information systems will emerge.

The Components of an Enterprise Risk Information System

Figure Risk Data Aggregation Target Architecture presents the component stack in the form used throughout this book. Five layers are distinguished, and each layer is annotated with the concerns it must address.



Figure 2.1 Risk Data Aggregation Target Architecture

Source Systems and Feeds

The base layer contains the systems of record and the external data services that supply the platform. Systems of record are operational: the core banking ledger, the trading and position platform, the policy administration system, the collateral management system and the credit risk engine. External feeds supply market prices, credit ratings, legal entity data and macroeconomic reference series.

Two properties of this layer determine much of the downstream architecture. The first is that source systems are optimised for their operational purpose and not for reporting. The core ledger is designed to post transactions and prove balances. It is not designed to expose a counterparty group exposure as at a historical date with a regulatory classification attached. The second is that source systems change on their own schedule, driven by operational requirements rather than reporting requirements. A schema change in a source system can silently invalidate a mapping that a reporting pipeline depends upon.

The architectural consequence is that the source layer must be treated as an external dependency rather than as a component under the reporting function's control. The relationship must be governed by an explicit interface contract that specifies the extraction mechanism, the timing, the expected volume range, the schema and the notification obligations for change. Chapter 7 develops the mechanics of that contract.

Ingestion and Integration

The second layer provides the mechanisms by which data leaves the source systems and enters the platform. The mechanisms are heterogeneous because the sources are heterogeneous. Where the source is a relational database under the institution's control, change data capture or direct extraction may be used. Where the source is a third-party system, an interface or an extract file may be the only available route. Where the source is a message stream, an adapter consumes and republishes.

The responsibilities of this layer are broader than movement. It must establish that a delivery is complete, that it corresponds to the period claimed, that it is attributable to a specific source version and that it can be replayed. These are control responsibilities, and they are the reason the ingestion layer cannot be treated as a purely technical concern. An ingestion mechanism that moves data reliably but cannot demonstrate completeness has not discharged its responsibility.

Quality and Control

The third layer contains the executable artefacts through which the institution asserts and verifies the quality of its data. It includes profiling capability, which discovers the characteristics of data; validation rules, which assert expectations about it; threshold definitions, which distinguish acceptable from unacceptable results; the exception queue, which receives and routes failures; and control reporting, which makes the aggregate result visible to management.

This layer is where the abstract vocabulary of data quality becomes operational. A dimension such as completeness is not a property that can be observed directly. It is measured by a rule that counts the records that should be present and the records that are present, expressed as a ratio, compared against a threshold, and reported. The rule is an artefact with an owner, a version and a history. Chapter 4 develops the design principles for such artefacts.

Reference and Master Data

The fourth layer holds the governed classifications, hierarchies and identities that give meaning to transactional values. It contains reference data such as country codes, currency codes, instrument classifications and product taxonomies, and master data such as legal entity identities, counterparty identities and customer records.

The layer is separated from the transactional layers because its behaviour is different. Transactional data is voluminous, largely append-only and rarely restated. Reference data is small, subject to in-place change and consumed by every downstream process. A single incorrect mapping in this layer propagates instantly and consistently to every consumer, which makes governance of the layer disproportionately valuable. Chapter 5 addresses it in detail.

Aggregation, Storage and Reporting

The fifth layer contains the stores in which data is held for reporting, the engines that compute aggregates and the mechanisms that render and transmit returns. It includes the enterprise warehouse, risk data marts, regulatory stores and immutable archives, together with the calculation, rendering and submission capabilities.

The distinguishing feature of this layer is that it must support reproducibility. A store that holds the current value of an attribute cannot reproduce a report produced six months ago, because the attribute may since have been restated. Reproducibility requires either immutable historical versions or the retention of sufficient inputs to reconstruct them. Chapter 9 develops the reconciliation framework that establishes whether the aggregates in this layer are correct, and Chapter 14 addresses the resilience properties that keep them available.

Supervisory Expectations as Architectural Requirements

The principles for effective risk data aggregation and risk reporting published by the Basel Committee in 2013 provide the most widely adopted articulation of what a risk information system must be able to do [49]. The principles are expressed as supervisory expectations rather than as technical specifications, which is both their strength and their practical difficulty. Their strength is that they remain applicable as technology changes. Their difficulty is that they leave the engineering translation to the institution.

Table 2.1 performs that translation for the fourteen principles grouped in the published document, expressing each as an architectural capability and an evidence artefact. The structure of the table is deliberate. A principle is not satisfied by a design decision; it is satisfied by a design decision plus a demonstration. An institution that has built a lineage capability but cannot produce a lineage report for a historical period has not satisfied the traceability expectation.

Table 2.1 Supervisory Risk Data Aggregation Principles Expressed as Architectural Capabilities and Evidence Artefacts

Principle Areas	Required Architectural Capability	Evidence Artefact
Governance	Documented data governance operating model with assigned decision rights	Council terms of reference, decision log, role descriptions
Data architecture	End-to-end lineage from source attribute to reported data point	Lineage model and data-flow register covering all material data
Accuracy and integrity	Automated validation and independent reconciliation of aggregated figures	Rule execution results, reconciliation statements and variance logs
Completeness	Controls asserting that all reportable items are included	Completeness metrics against expected population and control totals
Timeliness	Scheduled aggregation with monitored service levels	Run logs, service-level reports, exception and delay records
Adaptability	Tested capability to aggregate under stress and changed conditions	Scenario test evidence, capacity results and remediation plans

Three observations about this translation are worth stating.

First, the capability required by the governance principles is largely organisational rather than technical, yet it has a technical expression. A governance operating model assigns accountability for data domains. That assignment must be recorded somewhere and must be queryable, because an auditor will ask who owns a given attribute. The data catalogue discussed in Chapter 6 is the technical expression of the governance model.

Second, the accuracy and integrity expectations cannot be satisfied by validation alone. Validation asserts expectations about individual records. Reconciliation establishes relationships between aggregates. These are different control mechanisms and both are necessary. A record may pass every structural and value rule and still be wrong, because the rule set encodes the institution's current understanding and the understanding may be incomplete. Chapter 9 develops this argument.

Third, the adaptability expectations are commonly under-served. Testing that a platform can aggregate under stress, or that it can accommodate a new reporting obligation without redesign, is expensive and produces no visible benefit in normal operation. Institutions under delivery pressure defer it, and find during a stress event or a regulatory change that the claim of adaptability was never tested. Chapter 15 addresses how scenario testing can be built into the implementation methodology rather than added at the end.

Beyond the Aggregation Principles

The aggregation principles are the most prominent supervisory expectation bearing on data architecture, but they are not the only one. Three others shape the design space and are treated in the chapters indicated.

Model risk management guidance, of which SR 11-7 is the most widely referenced articulation, requires that institutions manage the risk arising from models used in decision-making and reporting [53]. Where a reported figure depends on a model an expected credit loss estimate, a probability of default, a valuation adjustment the data architecture must support the model lifecycle requirements of versioning, input traceability, output validation and change control. The interaction between model management and data management is treated in Chapters 8 and 15.

Internal control frameworks, notably the framework published by the Committee of Sponsoring Organizations, establish expectations for the design, operation and documentation of controls over reporting [54]. The framework's distinction between entity-level controls, process-level controls and transaction-level controls maps naturally onto the layered architecture of this book, and the requirement that control operation be evidenced maps onto the evidence artefacts discussed throughout.

Prudential and statistical reporting frameworks impose format and content requirements that must be reflected in the presentation layer. The extent to which these frameworks are codified in machine-readable taxonomies determines how much of the presentation logic can be automated and validated automatically rather than reviewed manually. Chapter 12 addresses the consequences.

The Regulatory Data Architecture Blueprint

Figure Evolution of Regulatory Data Engineering presented the Blueprint at its highest level of abstraction. This section develops it in detail, treating each layer as a set of responsibilities with defined interfaces.

The Blueprint is organised around a principle that may be stated simply: data flows through layers of increasing semantic richness and control, and each layer's output is a qualified input to the next. Raw extracts are qualified by completeness and attribution checks before they become landed data. Landed data is qualified by structural and value rules before it becomes standardised data. Standardised data is qualified by referential and relational rules before it becomes reportable data. Reportable data is qualified by reconciliation before it becomes a submitted return.

The principle has an important consequence. A defect that is not detected in the layer where it originates is detected downstream, where the cost of diagnosis is higher because the defensible context has been lost. A missing counterparty identifier in an extract is trivial to detect at landing and expensive to diagnose after aggregation. This is the engineering expression of the widely observed phenomenon that the cost of defect remediation increases with the distance between introduction and detection, and it motivates the emphasis throughout this book on control as close to origin as is feasible.

Interfaces Between Layers

The Blueprint is only useful if the interfaces between its layers are specified. An interface that is described as the standardised layer is not specified. A specification must state the data contract, the completeness semantics, the control results that accompany the delivery and the version identification of the producers. The data contract states the schema, the types, the nullability and the domain of each exposed attribute. Where the source is an external system, the contract is a governed agreement rather than an internal convention. Where the source is internal, the contract may be enforced automatically by schema validation at the point of ingestion.

Completeness semantics state what it means for a delivery to be complete. For a periodic extract, completeness may be defined against a control total supplied by the source, against a count of expected records derived from an independent query, or against a trailing average range with a tolerance. For a streaming feed, completeness cannot be defined as a set, and must instead be defined as an expectation about latency and continuity. Chapter 7 develops these semantics. Control results accompanying the delivery are the mechanism by which the receiving layer knows what has already been verified. A delivery that arrives with an assertion 1,284,552 records extracted, control total 948,332,118.42, checksum verified allows the receiving layer to build on that assertion rather than repeat it. A delivery that arrives as a bare file of unknown provenance forces the receiving layer to treat every attribute as suspect. Version identification addresses a problem that is chronically under-specified in practice. A dataset is not merely a set of values; it is a set of values produced by a specific version of a source system, extracted by a specific version of an extraction routine, transformed by a specific version of a rule set and classified against a specific version of reference data. Chapters 6 and 8 develop the mechanism by which these versions are captured and bound to the data they produced.

Regulatory Data Domains and System Responsibilities

The Blueprint requires that the enterprise's data be divided into domains and that each domain be assigned to a system of record and an accountable owner. The division is not arbitrary. It reflects the observation that some data is created by an operational process, some is maintained as controlled reference information, and some is derived by computation. Table 2.2 sets out a domain decomposition that covers the data used in most prudential and financial reporting obligations, together with the system of record and the principal reporting consumer in each case. The table is presented as an illustration of a decomposition rather than as a universal model; institutions will differ in granularity depending on their structure and obligations.

Table 2.2 Regulatory Data Domains with Systems of Record and Principal Reporting Consumers

Data Domain	System of Record	Principal Reporting Consumers
Legal entity and group structure	Entity master data service	Consolidated financial and prudential reporting
Counterparty and customer identity	Master data management hub	Credit risk, exposure and conduct reporting
Financial instrument and product	Instrument reference service	Accounting, regulatory capital and disclosure reporting
Contract and account terms	Originating operational system	Balance sheet, income and maturity analysis
Transactions and movements	Ledger and trading platforms	Income, cash flow and turnover reporting
Balances and positions	Ledger and position platforms	Balance sheet and exposure reporting
Collateral and netting agreements	Collateral management platform	Credit risk mitigation and leverage reporting
Credit risk parameters and ratings	Credit risk engine	Capital adequacy and impairment reporting
Market data and pricing	Market data service	Valuation and market risk reporting

Data Domain	System of Record	Principal Reporting Consumers
Classifications and taxonomies	Reference data hub	All reporting obligations

The domain decomposition performs three functions. It assigns accountability, so that a question about the correctness of a classification has a named owner rather than an escalation path. It establishes the boundary of a system of record, so that reconciliation between consumers and the authoritative source has a well-defined basis. And it informs the integration architecture, because the mechanisms by which a domain is exposed depend on whether the authoritative system can be queried directly. A common and costly failure is to leave the decomposition implicit. When domains are not defined, the same conceptual entity acquires multiple systems of record that disagree. The legal entity hierarchy may be maintained in the consolidation system, in the risk system and in the reporting system, each with slightly different contents. Reconciling three hierarchies is substantially more expensive than governing one.

The Enterprise Data Model

The domain decomposition has a counterpart at the level of structure: the enterprise data model. A data model in this context is not a physical schema. It is an abstraction that names the entities about which the enterprise holds data, the relationships between them and the attributes that qualify them. Three properties of the enterprise data model matter for regulatory reporting.

The first is that entity identity must be explicit and stable. A legal entity must have an identifier that does not change when the entity's name changes, that does not change when the entity is reorganised, and that can be mapped to the identifier schemes used by external authorities. The legal entity identifier standard provides one such scheme and is increasingly required for regulatory exchange [43]. Where an institution maintains its own internal identifier, the mapping between internal and external schemes is itself controlled reference data requiring governance.

The second is that temporal validity must be modelled explicitly. A relationship between a counterparty and a group is valid over a period. A classification of an instrument is valid over a period. If the model records only the current state, historical reporting becomes impossible to reproduce, and the temporal dimension is the one most commonly omitted. Bitemporal modelling, in which the period during which a fact was true and the period during which the institution believed it to be true are recorded separately, is the standard architectural response.

The third is that the model must accommodate the granularity required by the most demanding obligation. A model that stores exposures at entity level cannot satisfy an obligation that requires instrument-level disclosure, and recovering the lower granularity after the fact is generally impossible. This argues for modelling at the finest granularity the institution can economically maintain, and aggregating upward for reporting, rather than modelling at the coarsest granularity that satisfies a current obligation.

Architectural Patterns and their Selection

The Blueprint can be realised through several patterns, and the choice between them is among the most consequential decisions an institution makes. The centralised pattern concentrates all integration, quality and aggregation in a single governed platform. Its principal advantage is control: one lineage model, one rule set, one reconciliation framework. Its principal disadvantage is coupling. Every new source and every new obligation is a change to a shared asset, and the platform becomes a coordination bottleneck.

The federated pattern distributes integration and some aggregation to domain-oriented teams, with a central platform providing shared services and a central governance model providing coherence. The pattern has been articulated in the data mesh literature as a response to the coordination limits of centralised platforms, and its four organising principles domain ownership, data as a product, self-serve platform and federated computational governance map closely onto the concerns of this book [64]. Its advantage is scalability of change; its risk is divergence of semantic standards, which is precisely the risk that federated computational governance is intended to mitigate. The layered hub-and-spoke pattern, which is the pattern closest to the Blueprint as presented, uses a central conformance layer for identity, reference data and quality rules,

with decentralised transformation and reporting. It attempts to combine the control of centralisation with the change scalability of federation, at the cost of requiring a well-specified interface between the conformance layer and its consumers.

The choice between patterns should be driven by the institution's obligation profile and change rate rather than by technology preference. An institution with a small number of stable obligations and a low rate of source change is better served by centralisation and its simpler control model. An institution with many sources, many obligations and a high change rate will find that centralisation has become a delivery constraint, and that the governance investment required by federation is repaid. Chapter 16 develops the trade-off framework formally.

Non-Functional Requirements as First-Class Design Inputs

Regulatory data architectures are usually specified functionally: what data must be reported, in what structure, by when. The non-functional requirements determine whether the functional specification can be met reliably, and they are frequently left implicit until they are violated.

Six non-functional requirements recur and should be stated explicitly at design time.

1. **Reprocessing capability.** The platform must be able to re-execute any historical period on demand. The requirement is not merely that the data exist but that the code, rule set, reference data and configuration required to process it be recoverable. The measurable expression is the time to complete a full reprocessing of a specified historical window.
2. **Lineage coverage.** The proportion of reported data points for which complete column-level lineage can be produced. The requirement should be stated as a target with an approved exception list, because a target without an exception mechanism will be quietly abandoned.
3. **Determinism.** Repeated execution of the same period with the same inputs must produce byte-identical results for every reported figure. Failure of determinism destroys the ability to certify a reconciliation, because a reconciliation that passes once may fail on re-execution.
4. **Cut-off stability.** The data cut-off for each obligation must be a specification rather than an operational variable, and any deviation must be recorded with its quantified effect.
5. **Control execution evidence.** Every control invocation must produce a retained, queryable result record carrying the rule identity, its version, the population, the result and the threshold in force.
6. **Recovery objectives.** Per-obligation recovery point and recovery time objectives, stated in the terms the obligation uses rather than in platform terms.

Cost Ownership and the Economics of the Platform

A cost model, stated simply, clarifies several of this book's recommendations. The total cost of a regulatory data platform over its life has three components: construction cost, operation cost and change cost. Construction is incurred once, operation is incurred per reporting cycle, and change is incurred whenever a source, a framework or an obligation changes. The characteristic error of regulatory data programmes is to optimise construction cost. Extracts are written directly against the source, transformation logic is embedded in report code, validation is added at the point of consumption, and lineage is deferred. Each decision reduces construction cost and increases change cost. Because the first reporting cycle is the visible milestone and the change cost is borne years later by a different team, the error is rational at the level of the individual decision and irrational at the level of the programme. Making the economics explicit changes the conversation. A design decision that adds one month to first delivery and removes two months from each of the next ten cycles is, on those figures alone, favourable, and the argument can be made with numbers rather than with principles. The instrument for making it is the trade-off framework in Chapter 16, applied at the point of decision rather than ex post.

The Interface Contract Between Reporting and Source Systems

Because source systems are outside the reporting function's control, the relationship must be governed by an explicit contract. Six elements should be specified for every source. The delivery schedule and cut-off state when the data will be available and what period it represents. The completeness assertion states what the source will supply to allow completeness to be established independently, such as a control total or a row count. The schema and its change process state the interface structure and the notice period the source owner will provide before changing it. The quality

commitments state the levels of the attributes the reporting obligation depends upon, expressed as measurable expectations rather than as aspirations. The escalation path names the individuals on both sides who will resolve a delivery failure. The retention commitment states how long the source will retain the data required to correct or reprocess a historical period.

The last element is the one most frequently omitted and the most expensive to discover. Where a source system retains only the current state of an attribute, a historically incorrect value cannot be corrected at origin, and the institution must decide whether to correct it in the platform, which breaks the traceability chain, or to report it as it was, which requires an explanation.

Reading the Principles as Engineering Requirements

The supervisory principles for risk data aggregation are frequently treated as statements of management intent and are consequently cited at the level of the programme rather than the level of the design. Read as engineering requirements they are more precise, and each maps onto a measurable property of the platform [49], [51]. The requirement for completeness and accuracy maps onto the existence of an independent basis of comparison for every reported population and the operation of a control that asserts agreement with it. The requirement for timeliness maps onto a cut-off specification and a measured elapsed time from data availability to submission. The requirement for adaptability maps onto the number of governed artefacts that must change when a new obligation or a new source is introduced, which is the architectural expression of change cost. The requirement for accuracy of reporting maps onto the validation and reconciliation framework. The requirement for clarity and documentation maps onto lineage coverage and metadata completeness.

Sequential implementation, the final principle, is the one most often misread as a project-management observation. It is an architectural constraint: the aggregation capability must be established before the reporting machinery that depends on it, because a reporting layer built ahead of a governed aggregation layer inherits the defects of the sources it reads and cannot correct them where they originate [52].

The Obligation Inventory as an Architectural Input

An obligation inventory is a list of every regulatory return the institution must produce, and its architectural value lies in four attributes recorded against each: the reporting framework and its taxonomy version, the reporting frequency and submission deadline, the data domains the return draws upon, and the identity of the accountable reporting owner.

An inventory so structured answers the questions that determine architecture. Counting the returns by data domain identifies where a governed aggregation layer will be reused rather than rebuilt. Counting by frequency identifies which obligations require intraday or near-real-time data and which do not, which prevents streaming capability from being specified platform-wide on the basis of a single obligation. Counting by deadline identifies the periods in which the platform's capacity is contested. And the owner field converts an architectural requirement into an organisational one, because a return with no named owner cannot be scheduled or controlled [36].

Why Requirements Volatility Should Shape the Design

Reporting frameworks change at a rate that is high relative to the rate at which the underlying businesses change. Taxonomies are revised, data points are added and retired, definitions are refined, and validation rules are tightened. An architecture that treats each return as a self-contained pipeline rebuilds its mapping logic on each revision; an architecture that separates the mapping specification from the pipeline code revises a specification and re-executes.

The separation is the single most valuable structural decision available, and it is cheap to make at design time and expensive to retrofit. It is also the decision most often deferred, because the first obligation can be delivered faster by embedding the mapping in the code, and the cost of the deferral is borne by the next revision rather than by the current delivery [19].

The Independent Aggregation Capability

Where the institution derives its reported figures by projecting them from the same operational systems that produce its management information, a defect in those systems appears simultaneously in both and is therefore invisible to any

comparison between them. The principle of independent aggregation requires that the reporting figures be produced by a path that can be compared against the operational path, either because the aggregation is performed independently or because an independent basis of comparison is maintained.

In practice the independent basis is rarely a second full calculation, which would be prohibitively expensive. It is more often a control total posted by the operational system into the ledger, a valuation produced by an independent model, or an extract taken from the system of record through a separate interface. The architectural requirement is that such a basis be identified for each material reported quantity and that the comparison be operated as a routine control rather than commissioned as an investigation.

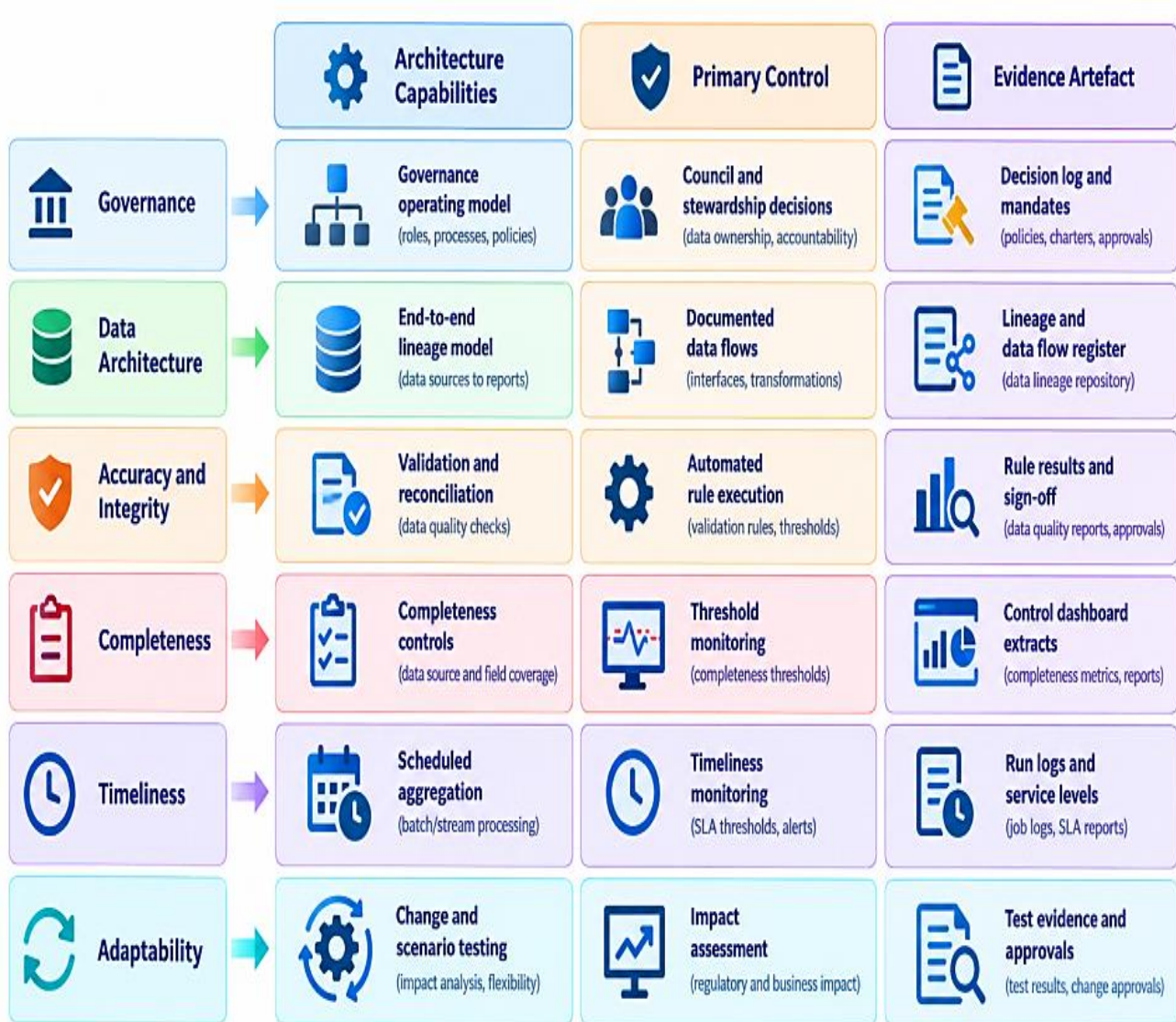


Figure 2.2 Risk Data Aggregation Principles Mapped to Architecture Capabilities

Figure Regulatory Data Domains and System Responsibilities sets out the resulting domain map, distinguishing the system of record from the principal reporting consumer for each domain and clarifying which domains are shared across obligations.

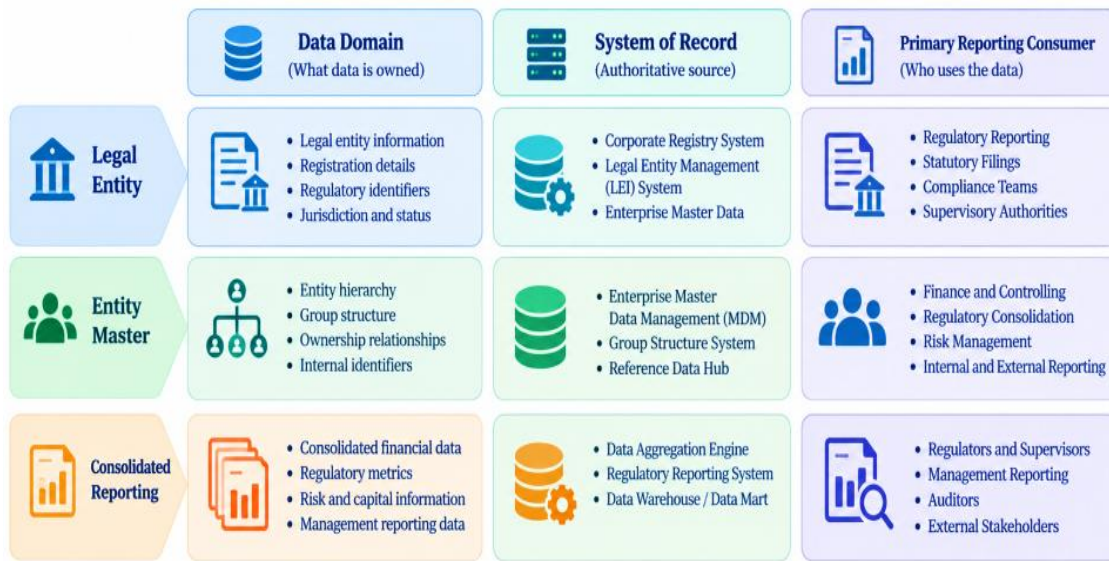


Figure 2.3 Regulatory Data Domains and System Responsibilities

The figure presents a seven-layer enterprise regulatory data architecture that manages the flow of information from source systems to regulatory reporting and submission. Layer 1, Source Systems, includes the core ledger, trading platform, and credit risk engine, which supply ledger extracts, position changes, and risk measurements. Layer 2, Ingestion and Integration, uses batch extractors, CDC adapters, and file and API connectors to collect data from these systems. Layer 3, Quality and Control, applies validation rules and data-quality checks, routing failed records to an exception queue while passing qualified data downstream. Layer 4, Reference and Master Data, provides entity hierarchies and classification services to support consistent entity mapping and regulatory classification. Layer 5, Aggregation and Storage, consolidates the information in an enterprise warehouse and risk data mart while maintaining historical snapshots in an immutable archive. Layer 6, Reporting and Submission, uses a reporting engine to generate reports and dashboards and a submission gateway to deliver regulatory filings. These layers work together to establish a structured and traceable reporting data pipeline.

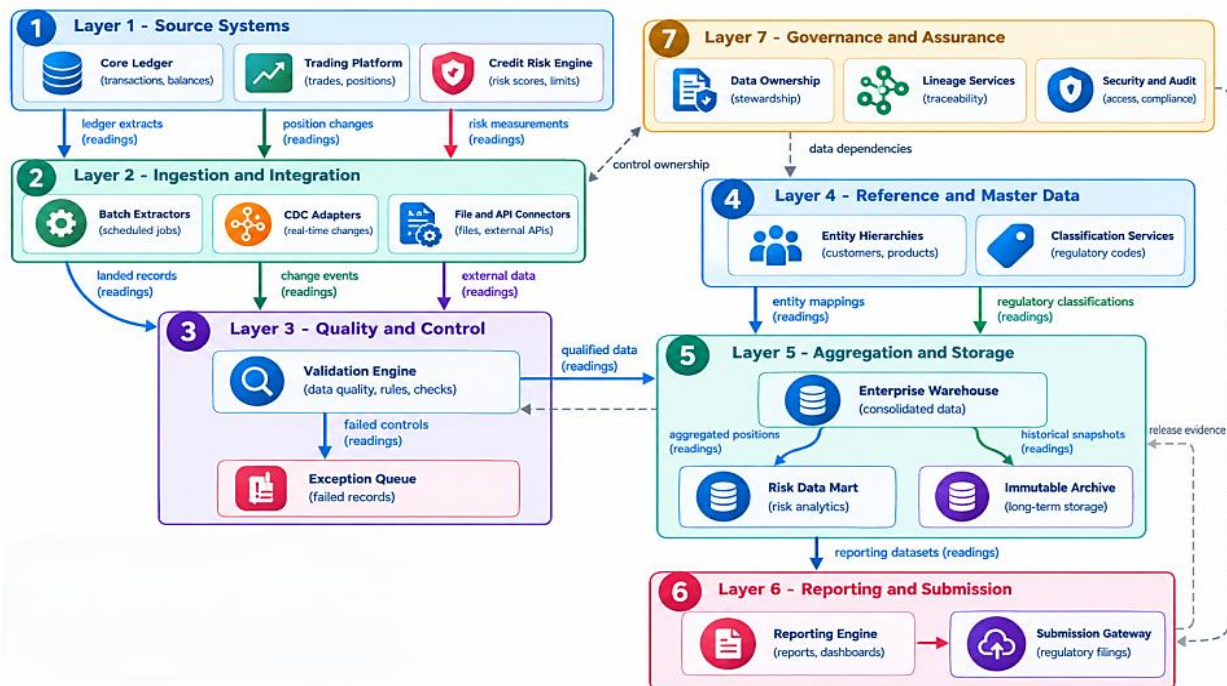


Figure 2.4 Seven-Layer Enterprise Regulatory Data Architecture for Governance, Quality Control, and Reporting

Layer 7, Governance and Assurance, provides data ownership, lineage services, and security and audit controls across the architecture. The connecting flows represent data dependencies, control ownership, reporting evidence, and the movement of qualified records between layers. By integrating governance, data quality, reference data management, storage, and submission controls, the architecture supports consistent regulatory reporting, improved traceability, exception management, and auditability throughout the data lifecycle.

CHAPTER 3

Data Readiness Assessment, Profiling, and Source-System Discovery

Before an institution can engineer a regulatory data pipeline, it must establish what data exists, where it originates and whether it is capable of supporting the obligation in question. This preparatory work is frequently compressed or omitted under delivery pressure, and the omission is repaid later in the form of surprises during testing. This chapter treats readiness assessment as an engineering discipline with defined stages, artefacts and acceptance criteria, and provides the techniques by which each stage is performed.

Data readiness is the extent to which the available data, in its existing condition and location, can support the intended use without unplanned remediation. The qualifier unplanned is essential. Any dataset can be made fit for any purpose given sufficient effort. The question that readiness assessment answers is not whether the data can be made fit, but whether it is fit now, and if not, what specifically must change and at what cost.

The distinction between readiness assessment and quality assessment is worth drawing, because the two are frequently conflated. Quality assessment evaluates data against defined expectations. Readiness assessment evaluates data against a prospective use, which may not yet have defined expectations. A file of counterparty records may be of high quality in the sense that its values conform to the expectations of the system that produced it, and simultaneously not ready for a reporting obligation that requires a legal entity identifier the file does not contain. Readiness is a relational property between data and purpose.

The Readiness Assessment Process

Figure Data Readiness Assessment Process presents the readiness assessment process as five sequential stages. The process is sequential in the sense that each stage consumes the output of its predecessor, but it is also iterative: findings at a later stage frequently require a return to an earlier one. A scope definition that omits a contributing source is discovered during profiling, and the inventory must be extended.

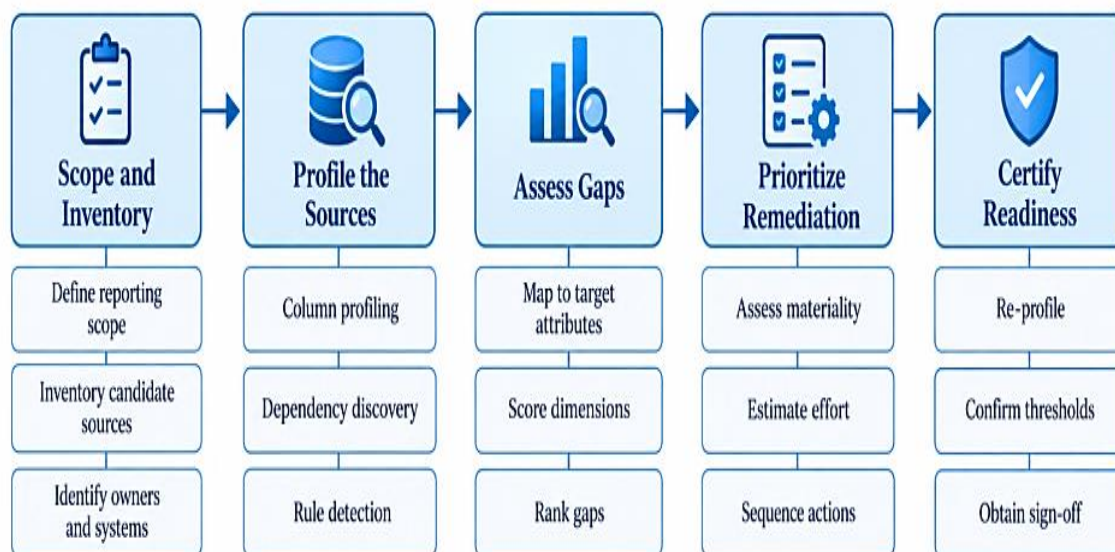


Figure 3.1 Data Readiness Assessment Process

The process is scoped to an obligation rather than to an enterprise. The reason is that readiness is relative to a purpose, and attempting to assess enterprise-wide readiness against all obligations simultaneously produces an unmanageable analysis. The practical approach is to perform readiness assessment once for each material new obligation, reusing the artefacts produced for prior obligations where the underlying sources overlap.

Stage One: Scope and Inventory

The first stage establishes what the assessment covers. It begins with the reporting obligation itself, decomposed into the data points it requires. A data point is the smallest unit of information that the obligation makes reportable: a specific value for a specific entity for a specific period. A prudential return may contain several hundred such data points; a large corporate reporting obligation may contain thousands.

The decomposition of the obligation into data points is a significant analytical exercise and is frequently underestimated. Where the obligation is published with a machine-readable taxonomy, the data points may be obtained directly from the taxonomy definition, together with their data types, permitted values and dimensional structure. Where the obligation is published as a specification document, the decomposition must be performed manually and reviewed by the reporting function.

With the data points established, the second element of the stage is the source inventory. For each data point, the assessment identifies the systems that could plausibly supply the required information. This is a hypothesis, not a conclusion. The hypothesis is tested during profiling and assessment. The inventory exercise is deceptively difficult for a reason that is worth naming: the institution's knowledge of its own data landscape is generally incomplete, and the incompleteness is invisible to the people who hold the knowledge. A system that was decommissioned two years ago may still appear in documentation. A system that was quietly introduced by a business unit may not appear at all. Techniques for mitigating this include reading actual data flows rather than documented ones, querying database catalogue views for tables containing relevant attributes, and interviewing operators rather than architects. The output of this stage is a source inventory artefact whose fields are given in Table 3.1. The fields are chosen to support the later stages directly: each field has a consumer in the assessment or remediation process.

Table 3.1 Source-System Inventory Template Fields and their Downstream Consumers

Fields	Purpose	Consumed by
Source identifier and name	Unique reference for all documentation	All stages and the lineage model
System classification	Distinguishes system of record from derived source	Domain decomposition and reconciliation design
Business owner and technical owner	Accountability for content and availability	Stewardship model and escalation paths
Extraction mechanism	Determines the ingestion pattern available	Ingestion design and failure-mode analysis
Timing and cut-off	Establishes the period semantics of the delivery	Period definitions and completeness controls
Expected volume range	Provides the basis for volume control limits	Completeness and anomaly detection
Key attributes supplied	Links the source to the data points it supports	Data-point mapping and profiling scope

Fields	Purpose	Consumed by
Schema change history	Reveals historically unstable interfaces	Impact assessment and testing scope
Known quality issues	Captures current institutional knowledge	Risk assessment and control prioritisation
Retention and availability	Constrains the ability to reprocess history	Reproducibility and archive design

The final field deserves emphasis. Where a source system retains only a short history, or has never retained the history at all, the institution's ability to reprocess historical periods is constrained by that fact and not by the reporting platform. Discovering this constraint during a remediation exercise rather than during assessment is a common and avoidable cause of delay.

Stage Two: Profiling the Sources

The second stage applies data profiling techniques to the candidate sources. Profiling is the systematic examination of a dataset to determine its structural and statistical characteristics without prior assumption about what those characteristics should be [7]. Figure Data Readiness Maturity Levels presents a taxonomy of profiling tasks, distinguishing single-column, multi-column, dependency and cross-source analysis. The taxonomy is useful because the techniques differ substantially in cost and in what they reveal, and an assessment programme should sequence them accordingly.

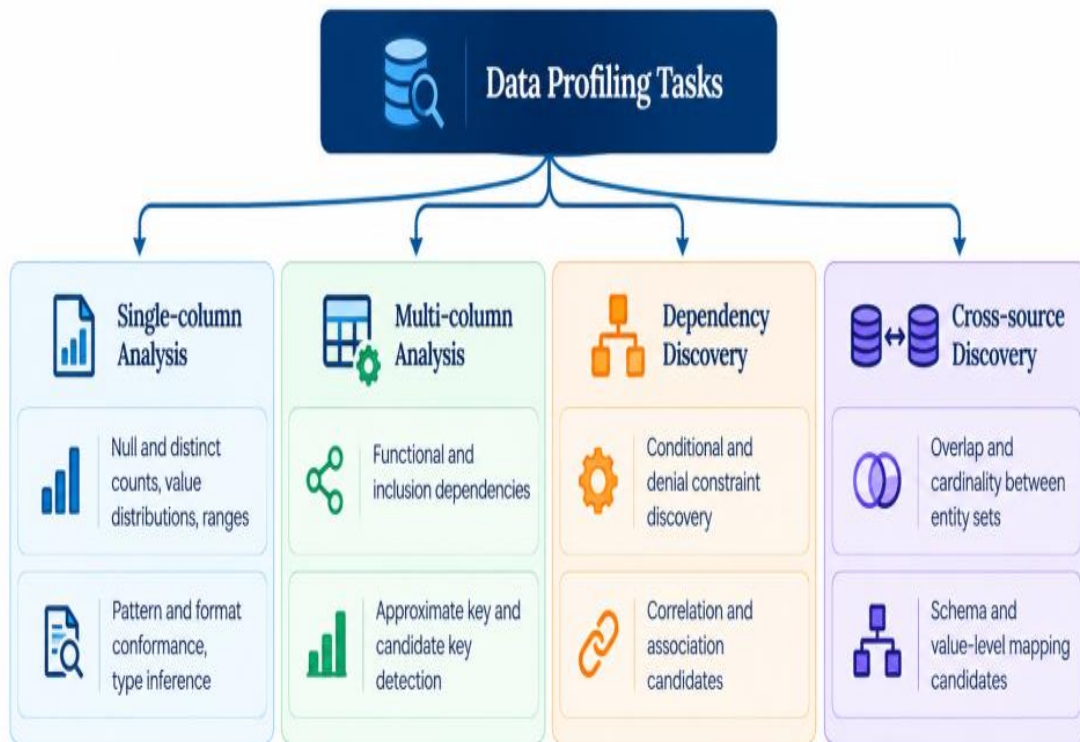


Figure 3.2 Taxonomy of Data Profiling Tasks

Single-column analysis examines each attribute independently. It computes null counts, distinct value counts, value frequency distributions, minimum and maximum values, length distributions and pattern conformance rates. The techniques are computationally inexpensive and reveal a great deal. An attribute that is populated in ninety-five per cent of records, where the missing five per cent correspond to a specific product type, tells the assessor something specific about the source's semantics.

Pattern analysis is particularly valuable for readiness assessment because it exposes representational heterogeneity that would otherwise surface as a defect during reporting. An identifier attribute may contain values in three different formats, reflecting three historical system versions or three input channels, with no field in the schema indicating which. Detecting the three formats during profiling allows the mapping rule to be designed before the pipeline is built rather than after it fails.

Multi-column analysis examines relationships between attributes within a record. Functional dependency discovery identifies attribute sets that determine other attributes, which is directly relevant to identifying candidate business keys and to detecting redundancy. Conditional dependency discovery identifies relationships that hold only within subsets of the data, which is relevant to discovering that a source system enforces different rules for different product categories.

Dependency and constraint discovery extends the analysis to data quality constraints that the data appears to satisfy. The discovered constraints are not statements about what the data should satisfy; they are statements about what it currently does satisfy. This distinction is critical. A discovered functional dependency in a source system is evidence about that system's behaviour, and it is a useful basis for a validation rule, but it is not a specification. Encoding discovered behaviour as specification permanently freezes whatever defects the source happened to contain at the time of profiling. Cross-source discovery examines relationships between datasets. It measures overlap between entity sets, which is the empirical basis for assessing whether two sources describe the same population. It proposes candidate join keys, which is the starting point for schema matching and for the identity resolution design discussed in Chapter 5.

Stage Three: Assessing Gaps

Profiling establishes what the data is. Gap assessment establishes the difference between what the data is and what the obligation requires. The comparison is performed at the level of the data point. For each data point, the assessment determines whether a source supplies the required value, whether it supplies it at the required granularity, whether it supplies it with the required timeliness and whether it supplies it with acceptable quality. Where the answer is negative, the assessment records a gap with a classification.

Gap classifications should distinguish at least the following four cases, because they imply entirely different remediation strategies.

A missing attribute gap arises when no source supplies the required information at all. Remediation requires either a new data capture at source, a derivation from existing attributes or an external acquisition. These are the most expensive gaps and the ones most likely to require project funding rather than configuration. A granularity gap arises when the information exists but at an aggregation level coarser than required. Remediation may be possible if the underlying detail exists in a source that is not currently extracted, which converts the problem into an ingestion scope question rather than a data capture question.

A semantic gap arises when the information exists but means something different from what the obligation requires. A source may hold a product classification designed for internal management, which does not map cleanly onto the regulatory taxonomy. Remediation is a mapping exercise, and its difficulty lies in the residual cases where no clean mapping exists the mapping is not a function but a relation with exceptions, and the exceptions require governed decisions. A quality gap arises when the information exists, at the right granularity, with the right semantic, but with a defect rate that would breach the reporting tolerance. Remediation may be at source, in the pipeline or both. Chapter 4 develops the assessment of quality gaps.

Stage Four: Prioritising Remediation

Gap assessment produces a list of deficiencies. Remediation prioritisation determines the order in which they are addressed, and this is fundamentally a risk-based rather than a technical exercise.

Three criteria are commonly used in combination. The first is materiality: the extent to which the gap affects the reported figures or their disclosures. A gap affecting a single immaterial line item is different from a gap affecting the completeness

of the counterparty population. The second is the presence of compensating controls: whether an existing control already mitigates the consequence of the gap, even if it does not address its cause. A gap in an automated validation may be substantially mitigated by an existing manual reconciliation. The third is remediation cost and duration, including the dependency on systems outside the reporting function's control. The combination of these criteria into a sequenced plan is the output of this stage. The plan identifies, for each gap, the remediation action, the accountable owner, the expected completion and the acceptance criterion. A plan that omits the acceptance criterion cannot be closed, because there is no defined condition under which the gap ceases to exist.

Stage Five: Certifying Readiness

The final stage re-executes the checks that the remediation was intended to address and issues a certification. The certification is a governed artefact. It states the scope of the assessment, the data points covered, the sources examined, the gaps identified, the remediation performed and the residual limitations, together with the name and role of the person accepting the residual limitations.

The last clause matters. Readiness assessments frequently conclude with residual gaps that cannot be closed before the obligation takes effect. The disciplined response is to document them explicitly and obtain acceptance from an accountable owner, not to redefine the assessment criteria so that the gaps no longer qualify. Institutional pressure to certify is considerable, and the certification's value depends entirely on its fidelity to the evidence.

Figure Source-System Discovery Map presents readiness maturity as five levels. The levels are useful for setting expectations with stakeholders, because an institution at level one cannot reasonably be expected to achieve level four within a single reporting cycle. The progression from undocumented sources to continuous assurance is a multi-year programme in most institutions, and the assessment's role is to establish where the institution currently stands and what the next level requires.

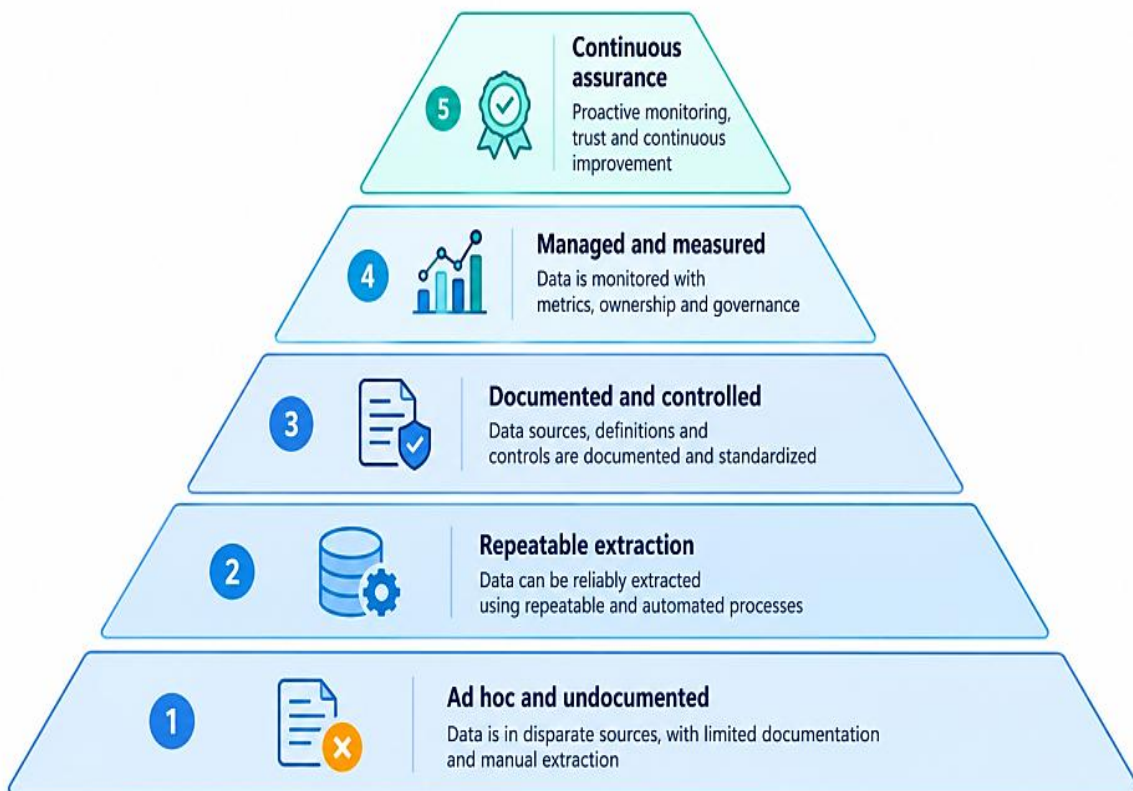


Figure 3.3 Data Readiness Maturity Levels

Source-System Discovery in Practice

Figure Taxonomy of Data Profiling Tasks presents a discovery map in the form frequently produced during assessment. The map records the candidate sources, the intermediate staging and conformance layers, and the reporting target, with the relationships between them made explicit.

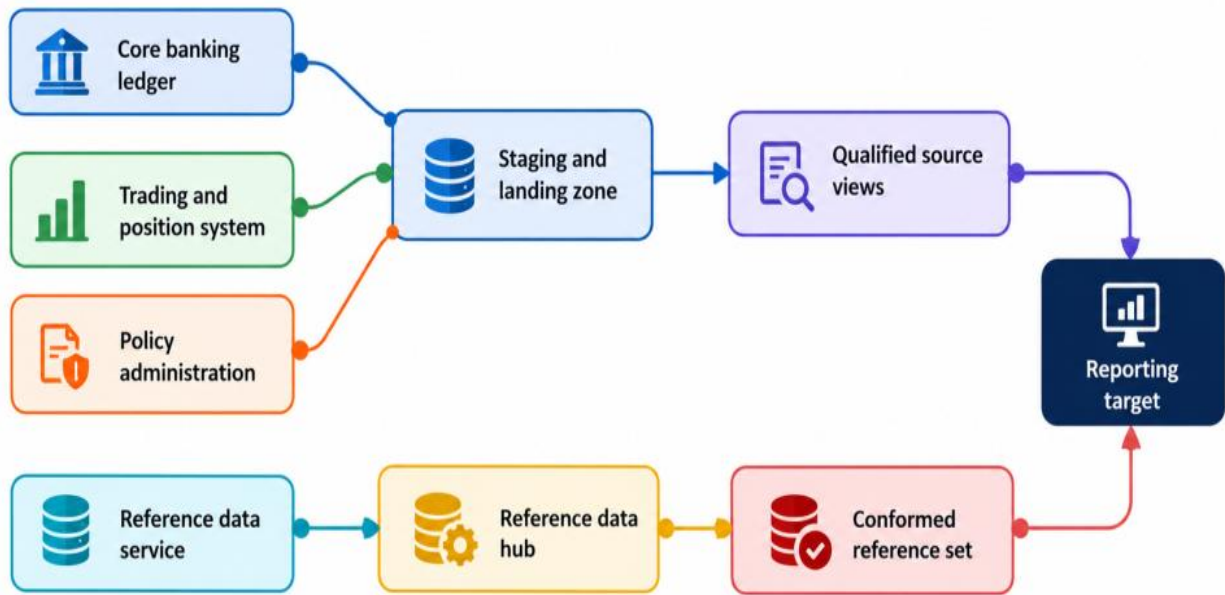


Figure 3.4 Source-System Discovery Map

The discovery map is a working artefact rather than a deliverable. Its value lies in the process of constructing it, which forces the assessment team to answer questions that documentation does not. Three techniques are particularly effective. The first is extraction-path interrogation. Rather than asking what data a system contains, the assessment asks how a specific reported figure was last produced. The answer identifies the actual extraction paths, which frequently differ from the documented ones. The technique is efficient because a small number of reported figures will typically traverse most of the contributing systems. The second is data-based discovery. Querying the data itself for evidence of origin reveals sources that documentation omits. Timestamp clusters reveal batch windows. Identifier formats reveal the systems that generated records. Null patterns reveal which attributes are populated by which upstream process. This is the practical application of profiling to discovery rather than to assessment, and it is frequently the technique that finds the source nobody remembered. The third is operator interview. The people who operate the pipelines know what the architecture documents do not: which jobs have to be run in a particular order, which feeds are known to be unreliable, which manual adjustments are performed each period. These facts are operationally critical and are rarely recorded. Capturing them during discovery converts tacit knowledge into documented control requirements.

Confirming the Period Semantics of Each Source

One discovery question deserves separate treatment because it is the source of a disproportionate share of reporting defects: what period does a delivery actually represent?

A file labelled with a date may represent positions as at the start of that date, as at the end, or as at some point during it. It may represent movements that posted during the date, which may differ from movements that were valued on that date. It may include or exclude transactions reversed later in the same period. The label does not answer the question, and the answer is frequently different for different source systems within the same institution.

The practical approach is to determine period semantics empirically for each source, by examining how the source's own totals reconcile to a known reference a ledger trial balance, a prior period's return, or an externally published figure. The result is recorded in the source inventory and becomes a specification input rather than folklore. This matters because period semantics errors do not produce validation failures. A pipeline that consistently uses the wrong cut-off produces a

complete, well-formed, internally consistent dataset that is wrong by one day's movement. The error is invisible to structural validation and is detectable only by reconciliation against an independent basis, which is why Chapter 9 treats reconciliation as an essential rather than optional control.

Readiness Assessment Artefacts and Their Maintenance

The readiness assessment produces four principal artefacts: the source inventory, the profiling results, the gap register and the readiness certificate. All four require maintenance, and the requirement to maintain them is frequently overlooked when they are created as project documentation. The practical maintenance mechanism is to bind each artefact to the platform that consumes it. The source inventory is not a document; it is a table in the metadata catalogue, updated when a source is added, changed or retired. The profiling results are not a one-off report; they are a recurring measurement whose history is retained, so that drift can be detected. The gap register is not a spreadsheet; it is a work queue with owners and ageing. The readiness certificate is a versioned record associated with a reporting obligation and a period.

Binding the artefacts to the platform has a secondary benefit. It makes the assessment repeatable. An institution that repeats readiness assessment for a new obligation can compare its findings against the previous assessment, which reveals whether the same classes of gap recur across the enterprise. Recurring gap classes indicate a systemic weakness at source rather than a series of unrelated local problems, and they justify investment in prevention rather than in repeated remediation.

Sampling Strategy for Large Source Systems

Profiling a source system with hundreds of millions of records raises a practical question: must the entire population be profiled, or is a sample sufficient? The answer depends on the profiling task. Set-based tasks null counts, distinct counts, minimum and maximum values, key uniqueness are computed exactly and their cost is proportional to the data scanned. Where the storage and compute permit, exact computation is preferable because a sampling error in a control denominator propagates into the control's reliability. Distributional tasks value frequency, pattern conformance, outlier detection are estimates regardless of whether they are computed on the full population or a sample, because the population itself is a snapshot that will change. For these tasks a stratified sample is adequate and substantially cheaper. The strata should follow the structure that matters to the obligation: by period, by product type, by entity category, so that a defect confined to one stratum is not diluted by the others.

Recording the Assessment Itself as Metadata

The artefacts of readiness assessment are most valuable when they become metadata rather than documents. The practical implementation records, for each source attribute mapped to a reported data point, four facts: the source and attribute, the profiling result at the time of assessment, the control that monitors it in production and the current status of any gap. Recording the assessment as metadata has three consequences. It allows the coverage question what proportion of reported data points have a profiled and controlled source? to be answered as a query rather than as a project. It allows the assessment to be repeated incrementally for a new obligation by reference to existing entries. And it makes the assessment's currency visible: an entry whose profiling result has not been refreshed in two years is an entry of uncertain value, and that uncertainty is itself recorded.

Judging Whether a Source Is Fit at All

Readiness assessment sometimes concludes that a candidate source is not fit and cannot economically be made fit. The judgement should be made explicitly and early, because continuing with an unfit source consumes the assessment's remaining effort and delays the discovery that an alternative is required. Three conditions justify the conclusion. The source does not contain the required information at any granularity, and the cost of adding the capture exceeds the cost of obtaining the information elsewhere. The source contains the information but with a defect rate so high that control would consume more effort than the value of the data, and no compensating control exists. Or the source's information is a derived artefact of a system that itself draws from a source that is available in better form, in which case the upstream source should be used. Where the conclusion is reached, it should be documented with its reasoning, because the same candidate source will be proposed again in a later assessment by someone who has not seen the earlier finding.

Structuring the Discovery Inventory for Reuse

The output of a discovery exercise is frequently a document, and a document has a limited life. The inventory is more valuable when it is structured so that its entries can be queried, extended and reused by a later exercise that shares part of its scope. The structural requirement is that the inventory be keyed at the level of the individual attribute rather than the system or the table. A system-level entry records that a system exists and supplies something; an attribute-level entry records what the attribute is called, what it means, what its permitted values are, who maintains it, how it is populated, how frequently it changes, what controls protect it and which reporting data points depend on it. The second granularity is what allows a question about a reported figure to be answered, because a reported figure depends on attributes rather than on systems.

The practical test of a well-structured inventory is whether it can answer three questions without further investigation: which attributes does this data point depend upon, which data points does this attribute feed, and where does the institution's knowledge of this attribute come from? An inventory that requires a fresh investigation to answer any of the three is a record of a past exercise rather than a reusable asset.

Profiling the Attribute Rather Than Only the Table

Table-level profiling row counts, key uniqueness, size and growth establishes that a table exists and behaves predictably. It does not establish that the attributes required for reporting are present, populated and conformant. The distinction matters because the requirement is almost always at the attribute level. An attribute-level profile records the proportion of null values, the distinct value count, the cardinality relative to the row count, the minimum and maximum where the domain is ordered, the frequency distribution of the most common values, the conformance of values to the expected pattern, and the proportion of values falling outside the declared reference set. Each of these is a potential control definition rather than merely a fact about the data, and the profile is most useful when it is recorded in the form in which the control will later be expressed.

The Elapsed-Time Dimension of Discovery

Discovery is often planned as a fixed-duration activity and then truncated when the duration expires. The truncation is more defensible when the work is sequenced so that the most consequential findings are produced first.

A sequencing that respects this principle proceeds from the reporting obligation inward. The data points of the highest-value obligations are decomposed first, because they determine which attributes must be found. Candidate systems are identified from those attributes rather than from a system inventory, which avoids examining systems that turn out to be irrelevant. Profiling then follows the attributes that are actually required. The result is that a discovery exercise truncated at any point has produced findings about the parts of the estate that matter most, rather than a complete description of whichever system was examined first.

Documenting What Was Not Found

An inventory records what exists. Its evidential value is increased substantially when it also records what was sought and not found, and what conclusion was drawn. Three negative findings deserve recording. The sought attribute that does not exist establishes that a requirement cannot be met from the current estate and that a new capture, a derivation or a proxy is required. The data of disputed meanings records that two systems use the same name for quantities that are not the same, which is the finding most likely to explain a later reconciliation difference. The candidate source rejected records a source that was considered and set aside, together with the reason, so that a later exercise does not repeat the evaluation. Negative findings are omitted because they are not artefacts of a system and appear to be conclusions rather than data. They are in fact the findings that prevent the repetition of work.

Reconciling Discovery with the Obligation

A discovery exercise should close by demonstrating that the attributes it identified are sufficient to produce every data point of the obligation it addressed. The demonstration is a mapping of obligation to attribute, and its value is that it converts a description of the estate into an assertion about deliverability.

Where the mapping is complete, the exercise has established that the obligation can be met from the existing estate, conditionally upon the controls identified. Where it is incomplete, the gaps are explicit: a data point with no identified source, an attribute with no identified owner, a period for which no historical data is available. These gaps are the input to the remediation planning described in Chapter 15, and they are far more useful when produced by the discovery exercise than when discovered during implementation. The figure illustrates an end-to-end framework for evaluating enterprise data readiness before data is used for regulatory processing and reporting. The Source Discovery stage identifies source systems, reporting obligations, and data ownership information. These inputs feed the Profiling Laboratory, where structural, statistical, and cross-source profiling techniques assess data coverage, distributions, and consistency across source populations. The profiling results provide evidence for the subsequent readiness assessment.

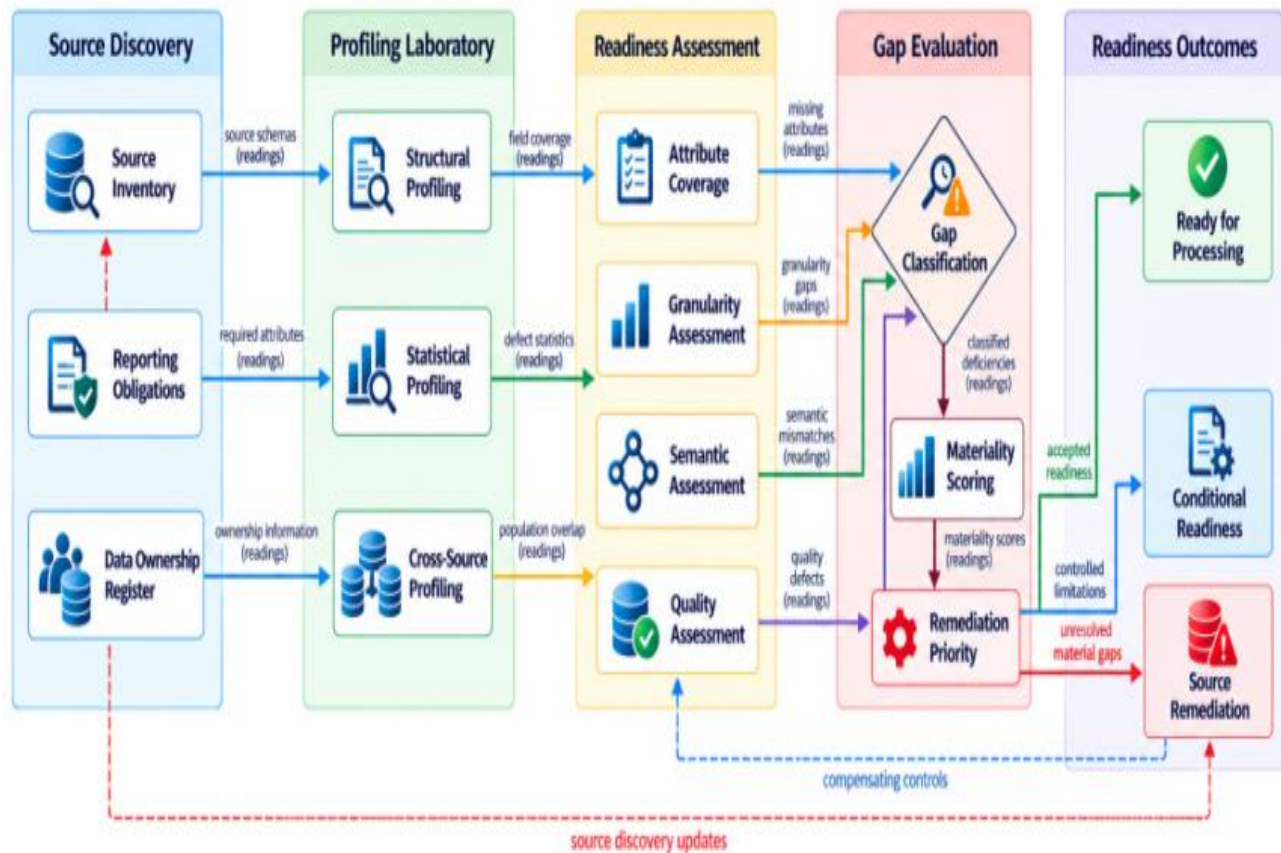


Figure 3.5 Enterprise Data Readiness Assessment and Gap Remediation Framework

The Readiness Assessment stage evaluates attribute coverage, data granularity, semantic consistency, and data quality. Its findings feed the Gap Evaluation stage, where identified deficiencies are classified, assessed for materiality, and prioritized for remediation. This process helps distinguish issues that may be managed through compensating controls from material gaps that require corrective action. The framework connects profiling evidence to specific readiness decisions rather than relying solely on subjective assessments. Finally, the Readiness Outcomes stage classifies the data as ready for processing, conditionally ready, or requiring source remediation. Accepted readiness and controlled readiness can proceed under the applicable conditions, while unresolved material gaps are directed toward remediation. The feedback loop allows remediation activities to update source discovery and subsequent assessments. Overall, the framework supports evidence-based readiness decisions, systematic gap management, and continuous improvement of enterprise data quality.

CHAPTER 4

Data Quality Dimensions, Validation Frameworks, and Control Engineering

Data quality is the property that determines whether a regulatory data platform can discharge its obligations. This chapter converts the abstract notion of quality into an engineering discipline: a set of dimensions that can be measured, a taxonomy of rules that can be executed, a framework that determines how rules are ordered and how results are interpreted, and a set of design principles that distinguish controls which prevent defects from controls which merely detect them.

The treatment begins from a position that is widely accepted in the information systems literature and less widely applied in practice. Data quality is not a single property. It is a multi-dimensional construct whose dimensions may conflict, and the relative importance of the dimensions depends on the use to which the data is put [1], [2]. A dataset optimised for completeness may sacrifice timeliness. A dataset optimised for accuracy may sacrifice completeness by rejecting records it cannot verify. The engineering task is not to maximise quality along every dimension but to establish, explicitly, the level of each dimension that the obligation requires, and to control to that level.

The Dimensional Structure of Data Quality

Figure Data Quality Dimensions presents the dimensional structure used in this book. It follows the widely adopted grouping of quality dimensions into intrinsic, contextual and representational categories, which distinguishes between qualities that inhere in the data itself, qualities that depend on the context of use, and qualities that concern how the data is presented [1].

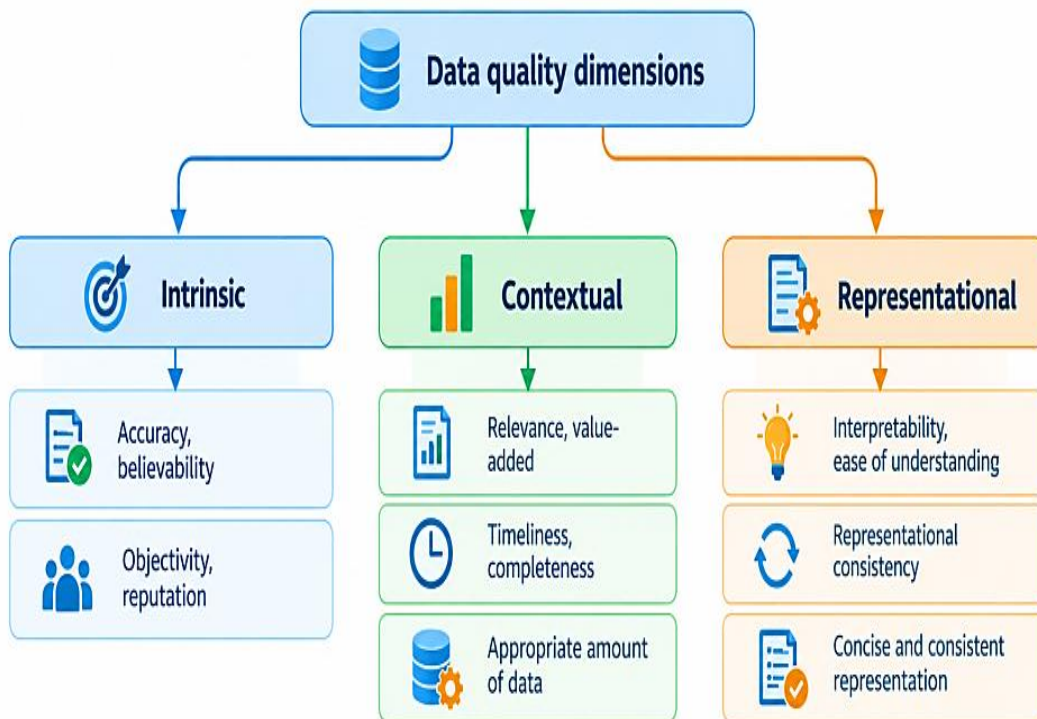


Figure 4.1 Data Quality Dimensions

Intrinsic Dimensions

Intrinsic dimensions are those that a consumer could assess without knowing the intended use.

Accuracy is the correspondence between a recorded value and the true value of the attribute it purports to represent. It is the dimension most commonly associated with data quality and the most difficult to measure, because establishing the true value requires an independent source of truth. In a regulatory context, the practical proxies for accuracy are reconciliation against an authoritative system and corroboration against an independent measurement of the same quantity.

Believability is the extent to which a consumer regards the data as credible. It is a subjective dimension and is therefore frequently dismissed in engineering discussions, which is a mistake. Believability determines whether a report is acted upon and whether a control is trusted. An institution in which the reporting function cannot persuade the business that a validation failure is genuine will find its controls circumvented by explanation rather than remediated by correction.

Objectivity is the extent to which the data is unbiased. In a regulatory context, the principal risk is discretionary adjustment: manual overrides applied without an evidence trail. Objectivity is maintained architecturally, by requiring that overrides be recorded with a justification, an owner and a retention period, rather than by prohibiting them.

Reputation and reputation of the source capture the accumulated institutional judgement about whether a particular system can be trusted. It is worth naming because it operates whether or not it is acknowledged. Trust in a feed is a form of prior probability that affects how anomalies are interpreted, and making it explicit allows it to be managed rather than merely inherited.

Contextual Dimensions

Contextual dimensions depend on the use to which the data is put, which is why they are the dimensions most directly relevant to readiness.

Relevancy is the extent to which the data bears on the question being asked. It is not a property of the data in isolation but of the data relative to a reporting obligation.

Timeliness is the extent to which the data is available when required, and it has two components that should be controlled separately: the latency of the data relative to the event it describes, and the lateness of its availability relative to the process that consumes it. A source that supplies positions with a two-hour lag may be entirely timely for a monthly return and entirely unusable for an intraday risk aggregation.

Completeness requires careful specification because it is the dimension most frequently mismanaged. Completeness is the ratio of the records present to the records that should be present. The numerator is observable; the denominator is not. Establishing the denominator requires an independent expectation of the population, obtained from a control total, a prior-period baseline, an independent query against a source of record, or a modeled expectation. A completeness control without an independent denominator measures something else usually volume stability and this distinction is among the most important practical points in control design.

Appropriate amount of data captures the requirement that a dataset contain neither more nor less than its purpose requires. Excessive granularity increases the volume to be processed, controlled and explained; insufficient granularity makes the obligation unservable. The dimension is a reminder that the granularity decision described in Chapter 2 is itself a quality decision.

Value-added is the benefit the data provides to its consumer relative to the cost of producing and controlling it. It is included here because it disciplines the proliferation of controls. A control whose result is never used adds operational cost and no assurance.

Representational Dimensions

Representational dimensions concern the form in which data is expressed.

Interpretability is the extent to which the meaning of the data is documented and discoverable. It is served by the metadata and glossary capabilities discussed in Chapter 6.

Representational consistency requires that the same concept be represented in the same way wherever it appears. This is the dimension that reference data governance (Chapter 5) primarily serves, and it is the dimension whose violation propagates most widely.

Concise representation requires that the representation be economical. Encoding a currency as a free-text description rather than a standardized code is a violation whose consequences appear as mapping defects downstream.

Ease of understanding concerns the fit between the representation and the capability of the consumer. In an automated pipeline, the consumer is a program, and the dimension collapses into representational consistency. In a supervisory or audit context, the consumer is a person reading a report, and the dimension becomes significant.

Reconciling Conflicting Dimensions

The dimensions conflict, and the conflicts must be resolved explicitly rather than by default. Three are common in regulatory data platforms. Completeness conflicts with accuracy where a validation rule rejects records that cannot be fully verified. A rule that requires a populated legal entity identifier will reject records that are economically material but whose counterparty has not yet been onboarded. The resolution is normally to admit the record with a quality flag rather than to reject it, so that the population is complete and the suspicion is explicitly recorded and subjected to downstream control.

Timeliness conflicts with completeness where a source is late. The resolution is normally to fix the cut-off and accept a partial population with a documented exception, rather than to delay the reporting deadline. This resolution is legitimate only if the exception is recorded, quantified and reconciled in the following period. Consistency conflicts with granularity where standardising to a common representation loses information that a specific consumer requires. The resolution is to retain the source representation alongside the standardised one in a provenance-preserving structure, rather than to discard it during standardisation. Chapter 8 develops the mechanics.

Validation as Executable Specification

A data quality dimension is a concept. A validation rule is the executable artefact that measures it. Figure Data Quality Control Loop presents the control loop through which rules operate: a rule is defined, executed, measured, evaluated against a threshold, remediated where necessary, reported and re-measured. The loop is closed, which is the property that distinguishes a control from a check.

Three design properties of validation rules determine whether they function as controls.

A rule must be declarative and versioned. A rule expressed as a fragment of procedural code embedded in a transformation job cannot be inventoried, cannot be reviewed in isolation and cannot be reused. A rule expressed as a declarative specification with a version identity can be inventoried, can be reviewed by a business analyst rather than only by an engineer, and can be associated with the period to which it applied. This last property is what makes historical reproduction possible: reproducing a report for a past period requires the rule set as it stood for that period, not as it stands now. A rule must have a defined population and a defined denominator. The population is the set of records to which the rule applies. The denominator is the expectation against which the rule's result is compared. A rule that counts null counterparty identifiers without stating how many counterparty identifiers there should be produces a number that cannot be interpreted.

A rule must have an owned threshold with a defined breach response. The threshold is the boundary between acceptable and unacceptable results. The breach response states what happens when the boundary is crossed: whether the pipeline

halts, whether records are quarantined, whether an exception is raised and to whom. A threshold without a response is documentation rather than control.

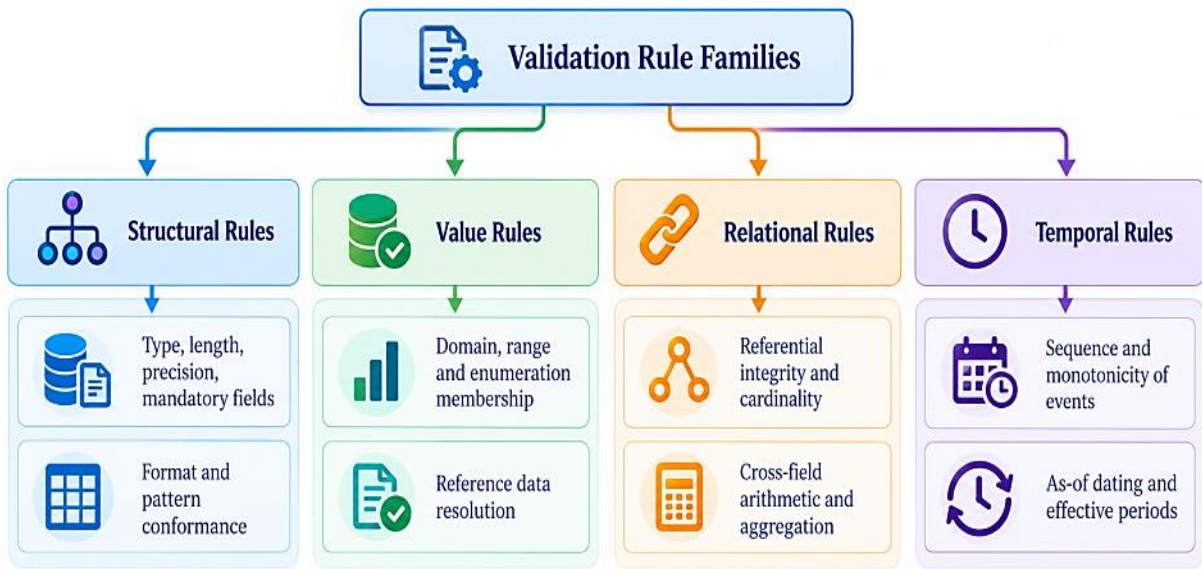


Figure 4.2 Data Quality Control Loop

A Taxonomy of Validation Rules

Figure Validation Rule Taxonomy presents a four-family taxonomy of validation rules. The families differ in what they can detect and in what they require to execute, which has direct implications for sequencing.



Figure 4.3 Validation Rule Taxonomy

Structural rules assert expectations about form: data type, field length, numeric precision, mandatory presence and pattern conformance. They are the cheapest rules to execute, they require no reference to other data and they are the natural first line of control. They are also the least informative, because they assert nothing about meaning. A record that satisfies every structural rule may be entirely wrong.

Value rules assert expectations about content: that a value falls within a domain, that it lies within a range, that it belongs to an enumerated set, or that it resolves against a reference dataset. Value rules require access to the reference data layer and therefore cannot be executed before reference data is available. Their dependence on reference data is a design constraint: a validation framework that executes before reference data has been loaded will silently validate nothing.

Relational rules assert expectations about relationships between records: referential integrity, cardinality, arithmetic consistency across fields and agreement between aggregate and detail. Relational rules are the most powerful family because they detect errors that no single record reveals. An aggregation rule that asserts that the sum of position-level exposures equals the entity-level exposure will detect omissions and duplications that satisfy every per-record rule.

Temporal rules assert expectations about time: the monotonicity of an event sequence, the validity of an effective period, the absence of gaps, and the coherence of as-of dating. Temporal rules are essential in regulatory reporting and are frequently the least developed family in practice, because they require the temporal model described in Chapter 2 to be present. Two further observations about the taxonomy are worth stating. The families are not alternatives; a mature control framework contains all four, applied at different points in the pipeline. And the ordering is deliberate: structural rules should execute before value rules, which should execute before relational rules, because a rule that depends on the output of a cheaper rule should not be executed on data that would have failed it.

Control Engineering Principles

Figure Quality Control Execution Pipeline presents the execution pipeline through which controls operate, distinguishing for each stage the input it consumes, the output it produces and the controls it applies. The pipeline is presented as an architecture because the sequencing of controls is an architectural decision with measurable consequences.



Figure 4.4 Quality Control Execution Pipeline

Place Control as Close to Origin as Feasible

The cost of a defect increases with the distance between its introduction and its detection. A defect detected at landing is a rejected file and a re-extraction. The same defect detected after aggregation is an investigation, a restatement and a supervisory conversation. The principle follows directly, and it is frequently violated for reasons of convenience: validation is added at the reporting layer because that is where the reporting team works, even though the defects it detects originated three systems upstream. Applying the principle requires that validation capability be available at every layer, not only at the end. This is an architectural requirement on the platform, not merely a process preference.

Prefer Prevention to Detection, but Build Both

Controls that prevent a defect are more valuable than controls that detect it, for the reason just stated. Prevention at source means capture constraints, referential integrity enforcement and workflow validation in the operational system. Where the reporting function controls the source, prevention is achievable. Where it does not, prevention requires a governed requirement on the source's owners and a mechanism to verify that it has been implemented.

Where prevention is not achievable, detection is necessary. The design error is to treat detection as equivalent to prevention and therefore to decline the investment in source-side change. Detection at scale is more expensive than prevention in operation, because it must be executed on every run and its failures must be investigated.

Make the Denominator Independent

The most common structural weakness in regulatory data quality control is a completeness or accuracy measure whose denominator is derived from the same pipeline as its numerator. If the count of expected records is computed by the pipeline being tested, the control cannot detect systematic omissions, because the numerator and denominator fail together. The denominator must be established independently: from a source control total, from an independently executed query, from a prior period adjusted by an independently known movement, or from an external benchmark. Independence is expensive to arrange, and its cost is the reason it is commonly skipped. It is also the single change that most improves the diagnostic power of a control framework.

Distinguish Hard Stops from Soft Flags

Not every rule failure should halt the pipeline. A control framework that halts on every failure will be permanently halted, and its operators will disable it. A framework that never halts will propagate defects silently. The engineering resolution is to classify rules by their consequence and to associate behaviour with the classification.

Hard stops are appropriate where a failure makes the output meaningless: a file that will not parse, a control total that does not agree, a reference dataset that has not been loaded. Soft flags are appropriate where a failure degrades quality without invalidating the output: a small number of records with missing optional attributes, a modest increase in an exception rate. Soft-flagged records proceed with an attached quality indicator, which downstream processes and reviewers can act upon. The indicator must survive into the reporting layer, because its purpose is to allow a reviewer to form a view.

Retain Rule Results as Evidence

Rule execution results are evidence of control operation. In an audit or supervisory examination, the institution will be asked to demonstrate not only that controls exist but that they operated, on which dates, with what results and with what follow-up. Retaining the results requires the results to be persisted in a queryable form with the rule identity, the execution timestamp, the population, the result and the threshold in force. Retaining them only in log files is not sufficient, because log files are retained for operational rather than evidentiary periods.

Control the Controls

The rule set is itself an artefact requiring governance. Rules are added in response to incidents and are rarely removed. Over time the rule set accumulates duplicates, contradictions and obsolete rules referring to superseded framework versions. The consequences are operational cost and, more seriously, loss of signal: a rule set with a persistent background of expected failures makes genuine failures invisible.

The governance mechanism is a periodic review in which every rule is classified as current, superseded or redundant, with the classification recorded and the removal of redundant rules approved. The review interval should be tied to the framework's own change cycle.

Threshold Design and Tolerance Management

Thresholds convert measurements into judgements, and their design deserves explicit attention because poorly designed thresholds produce either alarm fatigue or false assurance.

A threshold should be expressed in the unit in which the underlying risk is borne. A completeness threshold expressed as a percentage of records is appropriate where the consequence is proportional to the number of affected records. A threshold expressed as an absolute monetary amount is appropriate where the consequence is proportional to value. Where both matter, two rules are required. A threshold should be derived from an identified tolerance rather than from observed behaviour. The common practice of setting a threshold slightly above the recent worst result inverts the logic: it guarantees that the control will not fire on any deviation the institution has already experienced, including one that was unacceptable. The correct derivation begins from the consequence the institution is prepared to accept and works backward to the measurement. Thresholds should be tightened intentionally over time. An institution beginning control implementation against a legacy population may set thresholds that reflect current performance. Those thresholds should be time-limited and associated with a remediation plan, so that the control's tolerance narrows as the underlying quality improves. Without this discipline, the initial permissive threshold becomes permanent.

Handling Threshold Breaches

The response to a breach should be defined before the breach occurs, because designing a response under delivery pressure produces inconsistency. The defined response should specify who is notified, within what time, what the immediate treatment of the affected data is, what analysis is required, and what authority may approve a release notwithstanding the breach.

The last element is the most sensitive and the most necessary. Regulatory deadlines are immovable. There will be occasions when a breach cannot be remediated before the deadline. The institution must release with an exception, and the exception must be approved by an identified authority with the consequence of the breach explained. A framework that provides no such path will have it improvised, and improvised exceptions are the ones that fail audit.

Measuring the Control Framework Itself

A control framework is an engineering system and is subject to the same requirement for measurement as any other system. Four measures are useful.

- **Coverage** is the proportion of reported data points that are subject to at least one control. A data point with no control is an unassured figure. Coverage is the most basic measure and is frequently surprising when first computed.
- **Signal ratio** is the proportion of rule executions that result in a genuine exception. A low signal ratio indicates that the rule set contains rules that fire routinely on acceptable data, which desensitises operators. A very high signal ratio indicates that the rule set is too permissive or that the underlying data has deteriorated.
- **Mean time to remediate**, measured separately for detected exceptions and for exceptions identified by other means, indicates the operational responsiveness of the exception process described in Chapter 10. Exceptions remediated quickly by other means indicate a gap in automated detection.
- **Recurrence rate** is the proportion of exceptions whose cause has been seen before. A high recurrence rate indicates that remediation has been applied to symptoms rather than causes, and is the strongest available evidence that the control framework is functioning as a detector rather than as a corrective mechanism.

Worked Example: Designing a Completeness Control

The design sequence for a completeness control illustrates the principles of this chapter.

Consider a requirement to assert that every loan in the portfolio is included in the exposure calculation. The naive control counts the loans in the calculation and compares against the count of loans in the source extract. As Chapter 9 explains, this control is vacuous because both counts derive from the same path.

The corrected design proceeds in five steps. First, identify an independent denominator. The general ledger's loan control account holds the count of loan accounts and is posted by the operational system rather than by the extract, so it is a genuinely independent basis. Second, define the comparison precisely, resolving the differences that legitimately exist between the two populations: the ledger may include loans in a pending-disbursement state that are not yet reportable, and the extract may include off-ledger items that the ledger does not hold. Third, quantify those differences and encode them

as expected adjustments, so that the residual the control reports is genuine. Fourth, set the threshold from the institution's tolerance for unassured exposure rather than from the recent observed difference. Fifth, define the breach response, including which records are held, who is notified and what authority can release the return. The fifth step is the one that distinguishes a control from a measurement. A control determines an action; a measurement produces a number.

Calibrating Rules Against Labelled Data

Where a rule's behaviour must be calibrated rather than merely configured a near-duplicate detector, a plausibility threshold, an anomaly score the calibration requires labelled data: a set of records whose correct classification is known, established by manual review.

Two practical requirements apply. The labelled set must be representative, which normally means sampling across the strata that matter rather than taking the most readily available records. And it must be refreshed, because the distribution of the underlying data changes and a threshold calibrated two years ago is calibrated against data that no longer exists.

The calibration should record the false positive and false negative rates at the chosen threshold, and should record the cost asymmetry between them. Where a false negative permits a defective record into a report and a false positive consumes an hour of review, the thresholds should reflect that asymmetry rather than treating both errors as equally undesirable.

When to Retire a Control

A control should be retired when its assurance no longer justifies its cost. Four conditions justify retirement, and each requires evidence rather than assumption. The condition that no longer occurs: the source defect the rule was written to detect has been prevented at origin and has not recurred for a defined period. The condition that is now covered: a stronger control has been introduced that subsumes it. The condition that is now redundant: a reference data change has eliminated the population the rule addressed. The condition that is now misaligned: the rule asserts an expectation that the current reporting framework no longer requires.

In each case the retirement requires evidence of the justification and the approval of the rule's owner. A rule retired without evidence becomes a rule reintroduced at the next incident, and the cycle consumes the same effort repeatedly.

A Dimension-Level Control Register

The dimensions described in this chapter are most useful when each is associated with the control technique that operationalises it and with the failure mode that remains when the control is absent. Table 4.1 presents the register in that form. The register should be maintained as a governed artefact rather than as a reference table in a text: where a reporting obligation depends on a dimension, the obligation's control plan should name the register entry that protects it, and a data point whose controlling dimension cannot be named is a data point whose quality rests on assumption.

Table 4.1 Data Quality Dimensions with their Operational Control Techniques and Residual Failure Modes

Dimensions	Operational Control Technique	What Remains Undetected When the Control is Absent
Completeness	Control-total and population reconciliation against an independent basis	Records silently absent from any part of the delivery path
Accuracy	Re-performance against a source of record or rules-based recomputation	Correctly shaped values that are semantically wrong
Validity	Format, domain and reference-set conformance rules	Values that fit the pattern but not the framework requirement
Consistency	Cross-attribute and cross-system agreement checks	Mutually inconsistent values that each pass in isolation

Dimensions	Operational Control Technique	What Remains Undetected When the Control is Absent
Timeliness	Cut-off assertion with recorded data-as-of timestamps	Correct data that does not represent the reporting date
Uniqueness	Natural-key duplicate detection with survivorship rules	Double counting that inflates every downstream aggregate
Integrity	Referential checks against governed hierarchies and classifications	Orphan records that disappear from every aggregation
Traceability	Column-level lineage capture and rule versioning	Figures that cannot be reconstructed for a past period

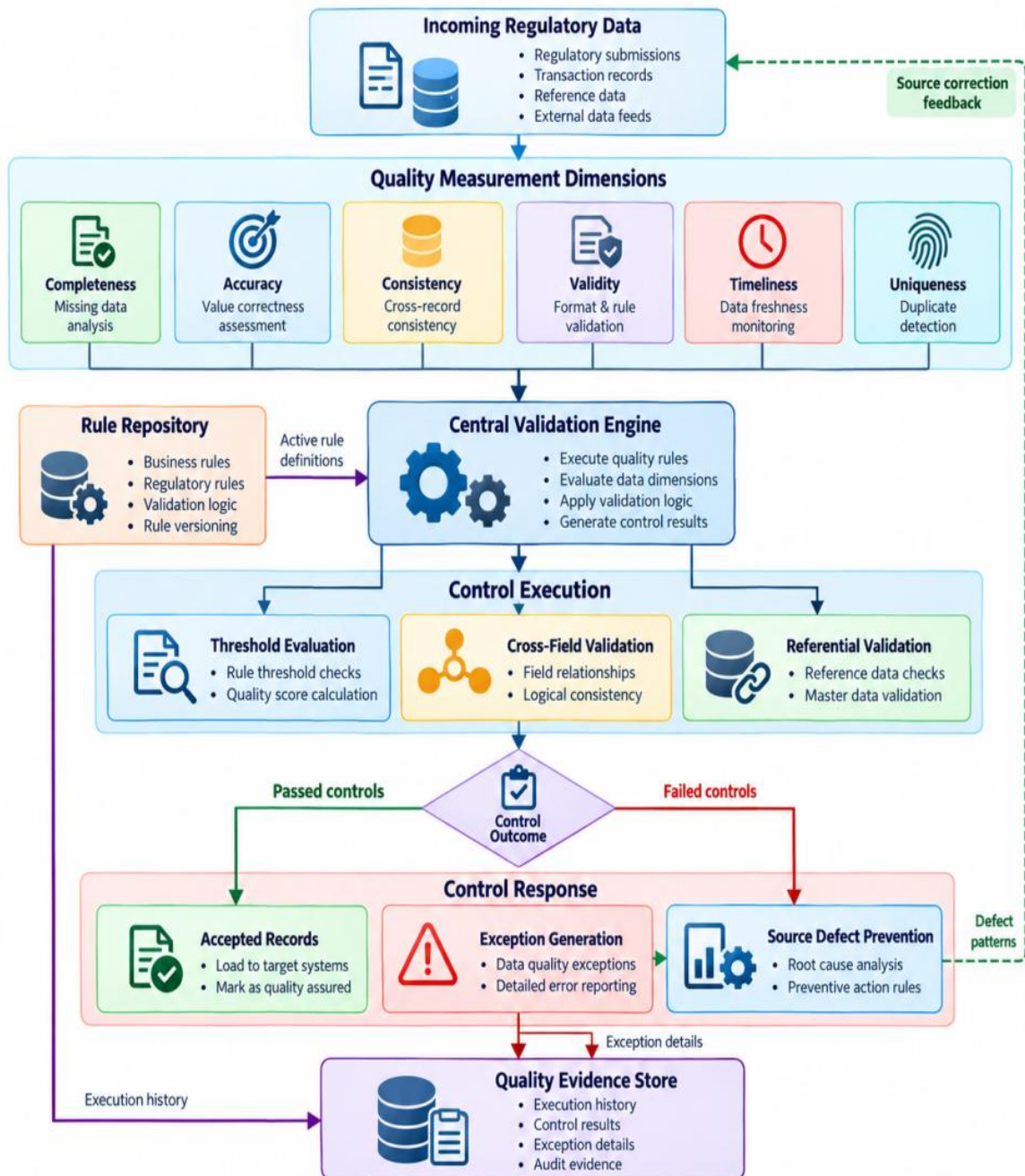


Figure 4.5 Enterprise Data Quality Measurement, Validation, and Control Framework

The figure presents an enterprise framework for measuring data quality, validating regulatory data, and managing control outcomes. Incoming regulatory data includes regulatory submissions, transaction records, reference data, and external data feeds. The Quality Measurement Dimensions assess completeness, accuracy, consistency, validity, timeliness, and uniqueness. These dimensions help identify missing values, incorrect information, cross-record inconsistencies, format violations, stale data, and duplicate records before the data proceeds through the validation process.

The Central Validation Engine executes business and regulatory rules obtained from the versioned Rule Repository. During control execution, threshold evaluation performs rule-based checks and quality-score calculations, cross-field validation evaluates relationships and logical consistency, and referential validation checks reference and master data. The resulting control outcome determines whether records pass or fail. Accepted records can be loaded into target systems and marked as quality-assured, while failed controls trigger exception generation and source-defect prevention activities, including root-cause analysis and preventive action rules.

The Quality Evidence Store preserves execution history, control results, exception details, and audit evidence to support traceability and review. Feedback loops return defect patterns and source corrections to the data-quality process, enabling recurring problems to be identified and addressed. Overall, the framework connects measurable quality dimensions with rule execution, controlled exception handling, preventive remediation, and evidence retention to support reliable regulatory data processing.

CHAPTER 5

Reference Data Management, Master Data, and Classification Standards

Reference data and master data occupy a small fraction of an institution's total data volume and account for a large fraction of its reporting defects. The reason is structural. Transactional data is largely append-only and records what happened. Reference data is small, changed in place, and consumed by every process that must interpret a transactional value. A defect in reference data does not affect one record; it affects every record that references it, consistently and silently. This chapter treats the governance, architecture and engineering of the layers that make transactional data meaningful.

The two terms require definition because they are frequently conflated. Reference data consists of the controlled sets of values used to classify and describe other data: country codes, currency codes, industry classifications, product taxonomies, instrument types, rating grades and status codes. It is characterised by being relatively static, externally or internally standardised, and shared across many consumers. Master data consists of the core entities about which the enterprise transacts: legal entities, counterparties, customers, instruments, contracts and organisational units. It is characterised by being large in population, subject to change, and requiring identity resolution across systems that were built independently. The distinction matters because the engineering responses differ. Reference data governance is primarily about change control, versioning and distribution. Master data governance is primarily about identity, survivorship and matching.

The Reference Data Lifecycle

Figure Reference Data Lifecycle presents the reference data lifecycle as a governed loop. The lifecycle is presented as a loop because reference data is never final: codes are added, deprecated and retired continuously, and each transition must be managed.

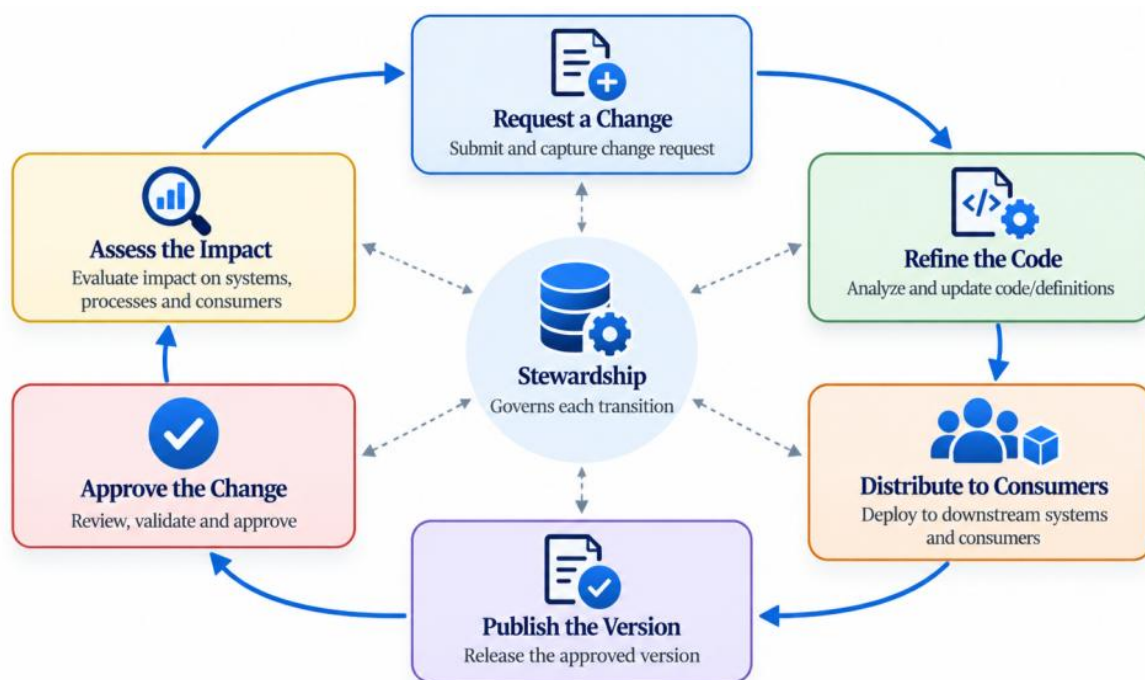


Figure 5.1 Reference Data Lifecycle

Every element of reference data should carry an effective date, a status and a version identity. The effective date states from when the value applies. The status distinguishes values that are active, deprecated and retired, so that a consumer can distinguish a value that should not be used from a value that should not exist. The version identity allows a consumer, or an auditor, to determine which version of the reference set was in force at a given time.

The retirement transition deserves specific comment because it is commonly unmanaged. A code is added, used in transactional data for a period, and then superseded. The naive response is to remove it, on the reasoning that it should no longer be used. The consequence is that historical data referencing the code can no longer be interpreted, and a report for a past period cannot be reproduced. The correct behaviour is to mark the code as retired while retaining its definition and its effective period permanently.

Impact Assessment as a Mandatory Control

Every change to reference data has an impact that extends to every consumer of the affected set. The architectural requirement is that the impact be assessed before the change is applied, and that the assessment be based on recorded relationships rather than on recollection. Determining the consumers of a reference set requires the metadata capability described in Chapter 6: the ability to traverse the relationship between a reference dataset and the columns that reference it. Where that capability exists, impact assessment is a query. Where it does not, impact assessment is an interview exercise whose completeness cannot be established, and the consequence is predictable: a reference data change is applied, a downstream mapping silently fails to resolve, and the defect appears as an unmapped-value exception in a reporting pipeline some time later.

The assessment should answer four questions. Which reporting obligations consume the affected values? Which transformation rules resolve against them? Which historical periods would be restated if the change were applied retroactively? And is there a transition period during which both the old and the new representation must be supported? The last question arises frequently and is frequently mishandled. A change in a classification standard for example, the introduction of a new industry sector definition characteristically applies to new data from a date, while existing data retains the old classification until it is restated. During that period the reference set must contain both definitions with non-overlapping effective dates, and every consumer must apply the correct one based on the date of the data rather than the date of the run.

Distribution and Caching

Reference data is consumed by many processes, frequently across multiple platforms. The distribution mechanism determines whether consumers see consistent values and whether changes propagate with controlled timing.

Two properties are required. The first is atomic publication. Consumers must not observe a partially applied reference data change, in which some values have been updated and others have not. Atomic publication is normally achieved by versioned publication: a new version of the set is published complete and consumers switch to it as a unit. The second is auditable consumption. It must be possible to determine which version of a reference set a given process used at a given time, which requires the version identity to be recorded in the process's own execution metadata rather than merely being present in the reference store.

Caching is unavoidable at scale and is the principal source of inconsistency. A consumer that caches a reference set reduces its dependence on the distribution mechanism and reduces latency, but a stale cache produces a value that was correct when cached. The architectural response is for caches to be version-aware: a cache entry is keyed by the reference version and is invalidated by version change rather than by elapsed time. This converts a timing-dependent inconsistency into a deterministic one that can be reasoned about and tested.

Classification Standards and Partial Adoption

Several reference data domains are governed by international standards, and conformance to those standards reduces the cost of integration and increases the credibility of reported data.

The legal entity identifier standard provides a globally unique identifier for entities that engage in financial transactions [43]. The currency code standard defines the coded representation of currencies [69]. The country code standard defines the coded representation of countries and their subdivisions [68]. The date and time standard defines representations for dates, times and intervals, including the treatment of time zones and durations [67]. The financial messaging metamodel provides a framework for defining message content standards across the financial industry [44].

Adoption of these standards within an institution is frequently partial, and the pattern of partial adoption is itself a source of defects. The common pattern is that the standard is adopted at the interface — the value transmitted in a regulatory submission or an interbank message — while internal representations remain proprietary. The consequence is a mapping layer at the boundary whose correctness determines the correctness of everything transmitted. That mapping layer is reference data in its own right and requires the governance described above.

A second common pattern is adoption of the standard's structure without adoption of its controlled vocabulary. An institution may use the country code format while using an abbreviated internal list that omits territories it does not currently operate in. The omission is invisible until a transaction in the omitted territory occurs, at which point the code resolves to nothing and the pipeline produces an unmapped value. The engineering response to both patterns is the same: treat the mapping between internal and standard representations as a governed reference dataset, subject to completeness control against the standard's own published code list. A completeness control that compares the institution's mapping table against the externally published standard detects omissions before they manifest as reporting defects.

Dimensional and Hierarchical Reference Data

Not all reference data is a flat list of codes. Many of the most consequential reference sets are hierarchies: the legal entity group structure, the product taxonomy, the industry classification with its several levels, and the organisational hierarchy. Hierarchies introduce two problems that flat lists do not have. The first is the consolidation path. Determining the total exposure to a group requires traversing the ownership hierarchy, applying ownership thresholds and treatment rules, and avoiding double counting through circular ownership. The traversal logic is not reference data but it depends upon it, and an error in the hierarchy produces an error in every aggregate that traverses it.

The second is the temporal validity of hierarchy relationships. Ownership changes. A subsidiary that was consolidated at 60 per cent ownership in one period may be consolidated at 100 per cent in the next. A hierarchy model that records only the current relationship cannot reproduce a historical consolidation. The bitemporal requirement introduced in Chapter 2 is most acute here.

The engineering response is to model hierarchies as relationships with effective periods rather than as parent pointers on entity records, and to require that every aggregation specify the date at which the hierarchy is to be evaluated. The specification is deliberate: it makes an implicit assumption explicit and reviewable.

Master Data Management Architecture

Figure Master Data Management Architecture presents the master data management architecture as five layers moving from raw identity evidence to governed golden records.

Source identity data consists of the records held by operational systems that describe the same real-world entities. The same corporate counterparty may appear in the lending system, the trading platform, the collateral register and the billing system, with different names, addresses and identifiers. The differences are not defects; they reflect the different purposes and the different onboarding processes of the systems that created them.

Standardisation transforms the raw representations into comparable forms. It includes parsing unstructured name and address fields into components, normalising case and punctuation, transliterating non-Latin scripts, expanding and contracting standard abbreviations, and applying address hygiene rules that validate and correct postal addresses against a reference source. Standardisation is more consequential than it appears. Because matching operates on standardised forms,

the quality of standardisation determines the achievable accuracy of matching. Two records for the same entity that are standardised differently will not match, regardless of the sophistication of the matching algorithm applied.

Matching and clustering determines which standardised records refer to the same entity. Chapter 5 continues this treatment below. Golden record construction applies survivorship rules to the attributes of the clustered records to produce the authoritative representation. Distribution exposes the golden records to consumers through a service interface and through conformed dimensional structures.



Figure 5.2 Master Data Management Architecture

The Golden Record and Survivorship

A golden record is the institution's authoritative representation of an entity, assembled from the records of multiple systems. It is important to be precise about what it is and is not. It is a governed composite, not a truth claim about the entity. It is correct in the sense that it is consistent with the institution's rules for constructing it, and those rules are what must be governed. Survivorship is the mechanism by which the composite is assembled. For each attribute, a survivorship rule determines which source value prevails. Table 5.1 presents the principal rule families and their appropriate application.

Table 5.1 Survivorship rule families for golden record construction

Rule Family	Selection Logic	Appropriate Where
Source precedence	Fixed ranking of source systems	Systems have clearly differentiated authority for the attribute
Most recent	Value from the most recently updated record	Attribute is genuinely time-varying and the latest change is authoritative
Most frequent	Value appearing in the majority of source records	Attribute is stable and sources are of comparable reliability

Rule Family	Selection Logic	Appropriate Where
Most complete	Value from the record with fewest null attributes	Records vary widely in richness
Longest or most specific	Value with greatest detail or specificity	Names, addresses and descriptions
Verified source	Value that has passed an external validation	Attribute can be validated against an external authority
Steward override	Value set by a data steward	Residual and disputed cases requiring judgement

Two design requirements apply to any survivorship scheme.

The first is that the rule must be attribute-level rather than record-level. A common simplification is to designate one source record as authoritative in its entirety. This is convenient and generally wrong, because the source most authoritative for the legal name is frequently not the source most authoritative for the registered address or the industry classification.

The second is that every source value must be retained. A golden record that overwrites the contributing values destroys the evidence required to explain a reported figure. Retaining the contributing values, with attribution to their sources, is what allows the institution to answer the question of why a reported counterparty name differs from the name held by a supervisor.

Entity Resolution

Figure Entity Resolution Pipeline presents the entity resolution pipeline as five stages: preparation, blocking, comparison, classification and clustering. The pipeline structure reflects the standard decomposition of the record linkage problem in the literature [12], [13].

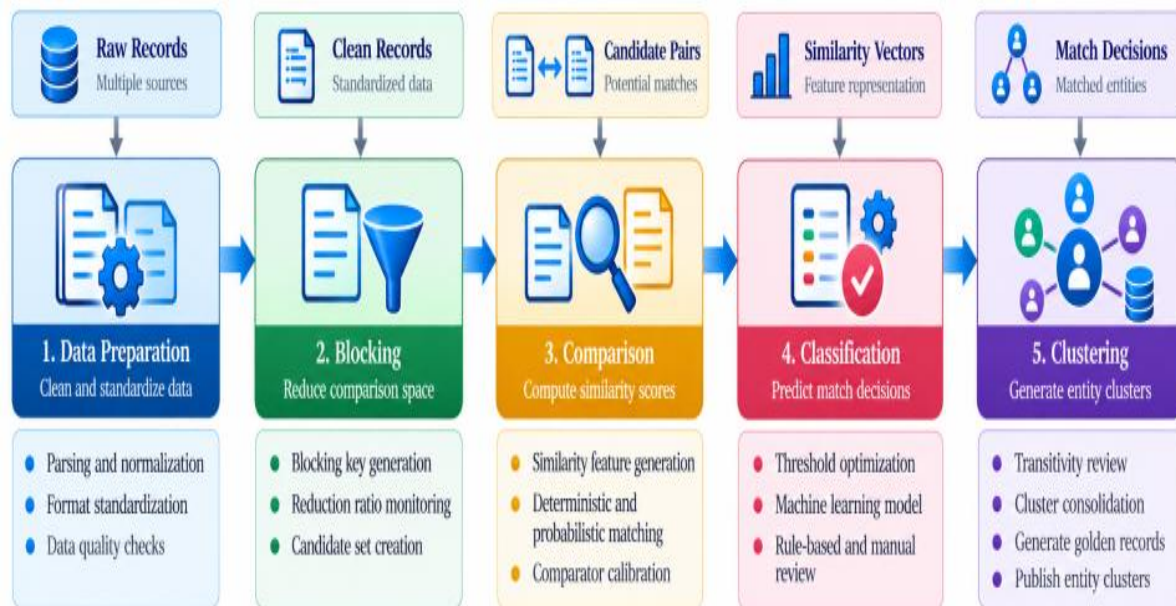


Figure 5.3 Entity Resolution Pipeline

Overview of the pipeline: the input is the set of standardised records from all contributing sources. Blocking reduces the comparison space from all pairs to plausible candidate pairs. Comparison computes similarity between the attributes of each candidate pair. Classification decides whether each candidate pair is a match. Clustering groups the matched pairs into entity clusters.

The Matching Problem and Its Cost Structure

The naive approach to matching compares every record against every other record. For a population of one million entities, this requires approximately five hundred billion comparisons, which is computationally infeasible for repeated execution. The entire architecture of practical entity resolution is therefore organised around reducing the comparison space without discarding true matches.

Blocking performs this reduction by grouping records into blocks according to a key and comparing only within blocks. The choice of blocking key is the central design decision, and it involves an unavoidable trade-off. A restrictive blocking key reduces the comparison space substantially but risks placing true matches in different blocks, producing false negatives. A permissive blocking key reduces false negatives but increases the comparison space.

Practical implementations use multiple blocking passes with different keys for example one pass on a normalised name prefix, one on a postal code, one on a national identifier where available and take the union of the resulting candidate pairs. Each pass catches matches that the others miss, at the cost of additional computation. The effectiveness of the scheme is measured by the reduction ratio, the proportion of the full comparison space that is examined, and by recall against a labelled sample.

Comparison and Similarity

Comparison computes, for each candidate pair, a similarity vector whose components measure the agreement of individual attributes. String similarity is computed by a range of measures with different sensitivities. Character-based measures are robust to transposition and minor misspelling but insensitive to reordering. Token-based measures handle reordering and the presence of additional tokens, which is common in entity names containing legal suffixes and trading names. Phonetic encodings handle spelling variation of names that sound the same. The literature on duplicate detection provides a comprehensive treatment of these measures and their comparative behaviour [12], and adaptive approaches learn the appropriate measure from labelled examples [18].

Numeric and date attributes are compared by exact match, by proximity with a tolerance, or by proximity with a decaying function that weights small differences more heavily than large ones. The choice depends on the attribute. A mismatch in a date of birth of one day is suspicious; a mismatch of ten years suggests different entities.

Classification and Threshold Setting

Classification converts the similarity vector into a match decision. The classical approach derives a match weight from the ratio of the probability of observing the agreement pattern among true matches to the probability among non-matches, an approach whose foundations are set out in the record linkage literature [14]. Modern implementations frequently use supervised classification, in which a model is trained on labelled pairs.

The central operational difficulty is threshold setting, and it is worth stating clearly because it is frequently mishandled. A single similarity threshold cannot simultaneously satisfy accuracy and recall requirements, because the distribution of similarity scores among true matches overlaps the distribution among non-matches. Raising the threshold reduces false matches and increases missed matches. Lowering it does the reverse.

The practical resolution is to use two thresholds rather than one. Pairs above the upper threshold are accepted automatically. Pairs below the lower threshold are rejected automatically. Pairs between the thresholds are routed to human review. The proportions flowing to each outcome are monitored, because a shift in the proportion requiring review indicates that the underlying data distribution has changed and that the thresholds require recalibration.

Clustering and Transitivity

Clustering groups matched pairs into entity clusters. The difficulty is that match decisions are made pairwise but must be aggregated into groups, and pairwise agreement is not transitive.

Consider three records: A, B and C. Suppose A matches B with high confidence, and B matches C with high confidence, but A does not match C. This arises where records are incomplete: A and C each match B on attributes that they do not share with each other. The practical options are to merge all three, to keep them separate, or to route the case to review. Merging on the basis of transitive closure is the simplest rule and produces the largest clusters. It also produces occasional catastrophes: a single erroneous match can chain together large numbers of unrelated records, and the resulting cluster is difficult to unpick once downstream systems have consumed it. This is the mechanism by which a single bad match produces a visible reporting anomaly affecting many entities. The engineering response is to compute clusters but to subject large clusters and clusters formed by long chains to review before publication, and to maintain the ability to split a cluster without disrupting downstream consumers. Maintaining that ability requires downstream systems to reference the entity by a stable surrogate identifier that can be reassigned, rather than by the attributes of the golden record.

Evaluating and Monitoring Matching Quality

Matching quality must be measured, and measuring it requires labelled data. The practical approach is to construct a reference sample of record pairs, labelled as matches or non-matches by a person with knowledge of the entities, and to compute precision and recall on that sample. The sample must be maintained, because matching performance degrades as the underlying data changes. New source systems introduce new name formats. New jurisdictions introduce new address structures. The periodic recomputation of precision and recall on a refreshed sample is the control that detects this degradation.

Two operational monitors complement the sample-based measurement. The match rate is the proportion of incoming records that match an existing entity. A sudden change in match rate indicates a change in the data or in the matching configuration. The review rate is the proportion of pairs routed to human review, and its behaviour indicates threshold drift as described above.

Master Data Governance in Practice

Master data management fails more often on governance than on technology. Three governance arrangements are decisive.

The first is the definition of the entity in scope. A programme that attempts to master every entity type simultaneously will stall. Successful programmes define a bounded entity type with a specific reporting motivation counterparties for credit risk reporting, for example and expand from there. The second is the resolution of competing authority. Where two source systems both claim authority for an attribute, the survivorship rule resolves the conflict technically but not organisationally. The underlying disagreement must be resolved by a named owner, because a survivorship rule built on an unresolved disagreement will be contested every time it produces a result someone dislikes. The third is the handling of the newly created entity. Master data management programmes focus on reconciling existing records and frequently under-specify the process by which new entities are created. Where entity creation occurs in an operational system and is propagated, the golden record inherits whatever defects the operational process permits. Where entity creation is centralised, it becomes a bottleneck. The practical arrangement is to define explicit creation paths with mandatory minimum attributes, so that a new entity enters the master data store with sufficient information to be matched and classified.

Worked Example: Resolving an Entity Across Four Systems

A corporate counterparty is onboarded by the lending business, then acquires a trading relationship, then posts collateral, then is billed for services. Four systems hold a record of the same entity, none of which knows about the others.

The resolution proceeds through the pipeline of Figure Entity Resolution Pipeline. Standardisation normalises the four names, which differ in the presence of a legal suffix, an ampersand versus the word and, and two transliterations of a non-Latin character. Blocking groups the four records by a normalised name prefix and national identifier. Comparison computes similarity across name, address, country and identifier, producing for each pair a similarity vector. Classification accepts the pairs above the upper threshold, rejects those below the lower and routes the residual to review, which in this case involves a single pair whose name similarity is moderate and whose address similarity is high because the entity has moved.

Clustering groups the accepted pairs. The published golden record selects the legal name from the lending system by source precedence, the registered address from an external verification source, the industry classification from the trading platform where it is maintained against a standard, and the tax identifier from the billing system where it was captured at onboarding. Each contributing value is retained with its source, so that when the counterparty asks why the institution holds a particular address, the answer identifies both the value and its provenance. The entity acquires a stable internal identifier that downstream systems reference, so that a subsequent correction to the golden record does not disrupt aggregation history.

Reference Data Change: A Managed Sequence

A change to reference data follows a sequence of ten steps, each of which is a control point.

The change is requested with a stated reason and a scope. It is logged in the reference data register with a unique identity. The impact is assessed against the lineage graph to identify affected rules, reports and periods. The consumers are notified with the effective date. The assessment is reviewed by the affected owners. The change is approved by the accountable owner of the reference set. The new version is built and validated, including a completeness check against the authoritative external source where one exists. The version is published atomically so that consumers see either the previous or the new version and never a mixture. Consumers are confirmed to have adopted the version, with non-adoption monitored rather than assumed. The superseded version is retained, with the code marked retired rather than removed. Steps four, nine and ten are the ones most frequently omitted, and each is the source of a characteristic defect: a consumer that fails silently for lack of notice, a consumer still resolving against a stale version months later, and a historical report that cannot be reproduced because the code it used no longer exists.

Figure Survivorship and Golden-Record Rules sets out the survivorship rule families applied when two contributing sources disagree, together with the attribute types for which each family is appropriate.



Figure 5.4 Survivorship and Golden-Record Rules

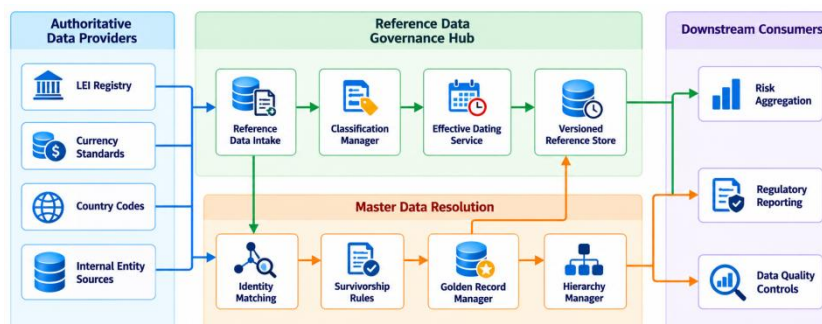


Figure 5.5 Enterprise Reference Data Governance and Master Data Resolution Architecture

An enterprise architecture for managing reference data and resolving master data across multiple authoritative sources. The Authoritative Data Providers layer includes the LEI Registry, currency standards, country codes, and internal entity sources. These providers supply foundational data to the Reference Data Governance Hub, where reference data intake,

classification management, effective dating, and a versioned reference store support consistent and controlled use of reference information across the enterprise.

The Master Data Resolution layer applies identity matching, survivorship rules, golden record management, and hierarchy management to identify duplicate or related entities and establish consistent master records. Identity matching links records that refer to the same entity, while survivorship rules determine which source attributes should be retained. The golden record manager maintains the consolidated entity representation, and the hierarchy manager organizes relationships among entities. Together, these components help maintain reliable and consistent master data.

The Downstream Consumers layer uses governed reference data and resolved master records for risk aggregation, regulatory reporting, and data-quality controls. Effective dating and versioned storage support the management of reference information over time, while centralized governance helps improve consistency and traceability across consuming systems. Overall, the architecture connects authoritative data providers, reference data governance, and master data resolution to support dependable enterprise risk and regulatory information processing.

CHAPTER 6

Metadata Management, Data Lineage, and Information Traceability

Regulatory reporting imposes a requirement that is unusual in data engineering: the ability to explain, after the fact, how a specific reported value was produced. This requirement cannot be satisfied by inspection of the data alone. It is satisfied by metadata and lineage. This chapter treats metadata as a first-class product of the platform rather than as documentation produced alongside it, and sets out the architecture, capture mechanisms and consumption patterns that make traceability a routine capability.

Metadata is data about data. The definition is accurate and unhelpful, because it encompasses almost everything the enterprise knows about its information assets. A more operational definition is that metadata is the information required to find, understand, trust and use a dataset. Under that definition, metadata includes the technical description of a dataset's structure, the business meaning of its attributes, the rules that produce it, the systems and processes that consume it, the controls applied to it, the people accountable for it, and its quality history.

The distinction between metadata as documentation and metadata as a platform capability is the central engineering point of this chapter. Documentation is produced at a point in time, drifts from reality as the systems it describes change, and is consulted by people. A metadata platform is populated by instrumentation, maintained as part of the change process, and queried by both people and programs. The traceability requirement described in Chapter 2 can be satisfied by the second and not by the first.

Categories of Metadata

Figure Metadata Catalog Architecture presents the metadata catalogue architecture as five layers from source to consumption. Before describing the architecture, the categories of metadata that it must hold require definition, because they are produced by different mechanisms and consumed by different audiences.



Figure 6.1 Metadata Catalog Architecture

Technical metadata describes the physical structure of data assets: schemas, tables, columns, data types, constraints, indexes, partitions and storage properties. It is largely discoverable by automated crawling of database catalogues and file system listings. Its principal value in a regulatory context is that it establishes what exists and where, which is the foundation of every subsequent question.

Operational metadata describes the execution of processes: which job ran, when it started and finished, how many records it processed, whether it succeeded, what resources it consumed. It is produced by the orchestration and execution platforms. Its value in a regulatory context is that it provides the evidence that a pipeline ran and that its behaviour was as expected. It is also the source of the freshness and volume signals discussed in Chapter 14.

Business metadata describes meaning: what a dataset represents, what an attribute means, how it is derived, what its permitted values are, who owns it and what quality expectations apply. It is produced by people, not by crawling, and this is the reason it is chronically incomplete. Its value in a regulatory context is that it allows a reported figure to be explained in terms that a supervisor or auditor can evaluate.

Lineage metadata describes relationships between data assets: which source column populates which target column, through which transformation, under which rule version. It is produced by parsing transformation code, by instrumenting execution, or by a combination. Chapter 6 develops it separately because of its centrality to traceability.

Governance metadata describes accountability and policy: which domain an asset belongs to, which owner and steward are accountable, what classification and handling requirements apply, what retention period is mandated, and whether the asset has been certified. It is maintained through workflow rather than discovered.

The Business Glossary and Its Relationship to the Catalogue

The business glossary and the data catalogue are distinct artefacts that are frequently conflated to the detriment of both. The glossary defines business concepts. It contains entries for terms such as exposure, counterparty, default, materiality and reporting period, each with a definition, a scope, a relationship to other terms, and an accountable owner. It is a terminology artefact. Its consumers are people, and its purpose is to eliminate the ambiguity that arises when the same word is used with different meanings in different parts of the institution.

The catalogue describes data assets. It contains entries for datasets, tables and columns, each linked to the technical and operational metadata that describes it and, where the institution has succeeded in the linking exercise, to the glossary terms that define its meaning. It is an asset inventory. Its consumers are both people and programs. The relationship between them is many-to-many and is the most valuable link in the metadata model. One glossary term may be implemented by many physical columns across many datasets; one physical column may implement several distinct concepts in different contexts. Establishing these links is the work that converts a catalogue from a directory of tables into a semantic map of the enterprise's data. The linking exercise is substantial, and it is common for it to be deferred indefinitely. The consequence of deferral is that the catalogue answers the question what columns exist in this table? well and the question where in our systems is the concept of exposure implemented? not at all. The second question is the one that regulatory traceability actually requires, which is why the glossary is treated here as a first-class artefact rather than as an optional enhancement.

Certified and Curated Assets

A mature catalogue distinguishes between assets that are merely catalogued and assets that have been certified as fit for a stated purpose. Certification asserts that the asset's definition has been agreed, its controls are defined and operating, its lineage is documented, and an accountable owner has accepted responsibility.

Certification matters architecturally because it gives consumers a signal they can act upon. A pipeline engineer selecting a source dataset can distinguish a certified dataset from an uncatalogued one, and can require justification for the latter. Without certification, every consumer performs its own assessment, which is expensive and produces inconsistent conclusions.

Lineage: Concepts and Granularity

Lineage is the record of the relationships through which data moves and changes. Its granularity determines what questions it can answer.

- **Table-level lineage** records that a target table is populated from a set of source tables. It answers questions about the dependency structure of the platform and supports impact analysis at the level of assets. It is obtainable in most environments at moderate cost.
- **Column-level lineage** records that a specific target column is populated from a specific set of source columns. It answers the question that regulatory traceability requires: which source attributes contribute to this reported value? It is substantially more expensive to obtain because it requires interpreting transformation logic rather than observing dependencies.
- **Row-level lineage** records that a specific target row derives from a specific set of source rows. It answers the most specific question: which records produced this number? It is the most expensive and is generally achieved by instrumenting execution rather than by static analysis.
- **Provenance** extends lineage with the derivation semantics: not merely which source contributed, but through which transformation and under what conditions. The distinction between why-provenance, how-provenance and where-provenance provides the vocabulary for this extension [27], [28].

Figure Column-Level Lineage Graph presents the three provenance notions mapped to the questions that regulatory and audit consumers actually ask.

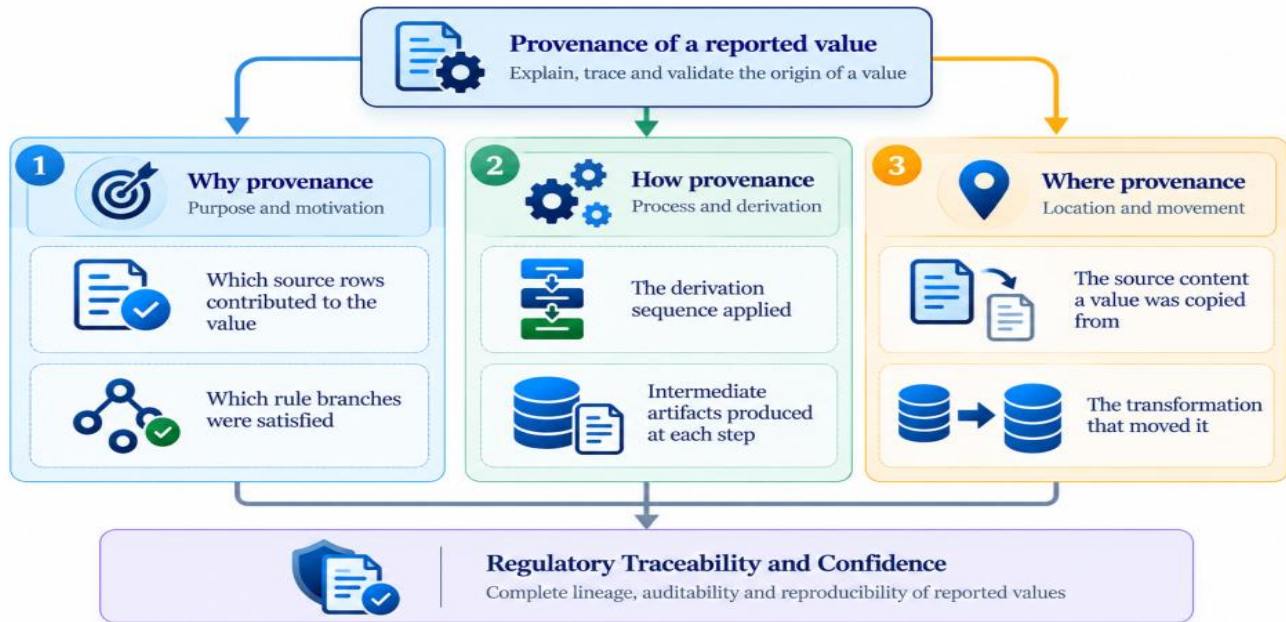


Figure 6.2 Provenance Model: Why, How and Where

Where-provenance answers where did this value come from? and corresponds to column-level lineage. Why-provenance answers which source records caused this result to be produced? and corresponds to row-level lineage with the derivation conditions attached. How-provenance answers through what sequence of steps was this value derived? and corresponds to the recorded execution of the transformation graph.

The practical significance of the distinction is that the three notions answer different supervisory questions and therefore impose different architectural requirements. An institution that captures only column-level lineage can answer the question which source column feeds this reported cell? but cannot answer which transactions produced this total? An institution that captures row-level provenance can answer both, at substantially higher cost.

Capturing Lineage

Figure Provenance Model: Why, How and Where presents a column-level lineage graph for a reported exposure figure, traced from four source attributes through staging, transformation and aggregation to a specific report cell. The graph is the artefact that a supervisor or auditor is shown when they ask how the figure was produced.

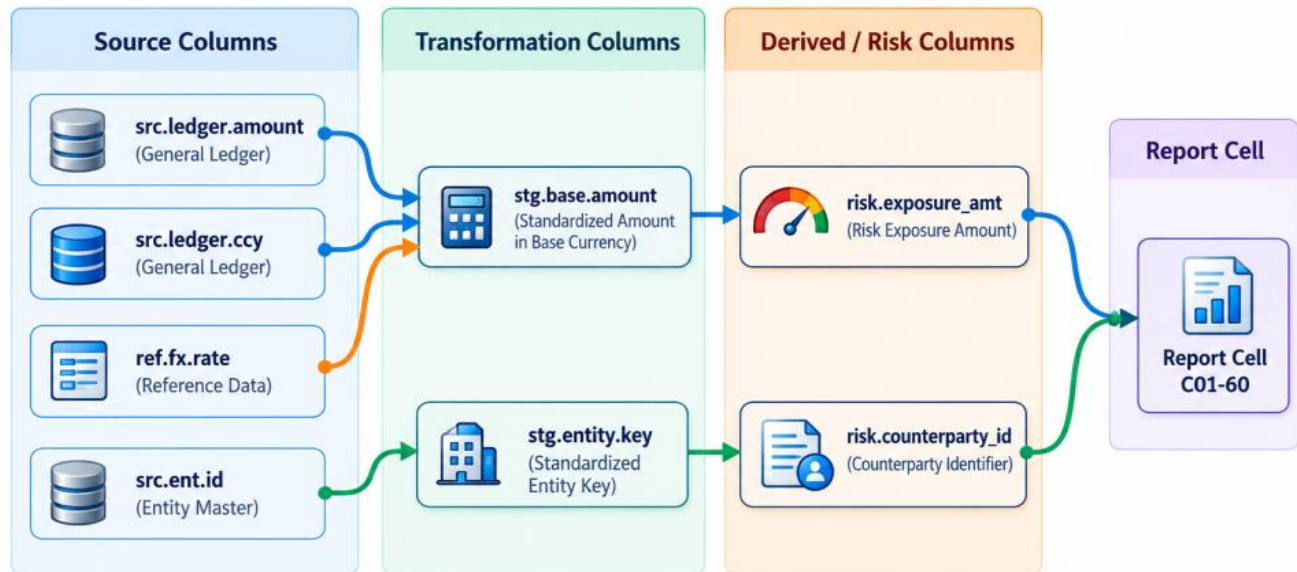


Figure 6.3 Column-Level Lineage Graph

Three mechanisms produce lineage, and mature implementations use all three because each has characteristic blind spots.

- **Static analysis** parses the code, query text or mapping definitions that express transformations and derives lineage from their structure. Its advantage is that it captures the intended relationship expressed in the logic. Its limitations are equally important: it cannot see runtime behaviour, it fails on dynamic constructs whose targets are determined at execution time, and it reports lineage that may never be exercised in practice.
- **Execution instrumentation** captures what actually happened when a transformation ran: which inputs were read, which outputs were written, which rules were applied. Its advantage is that it is ground truth for the execution. Its limitations are that it captures the actual rather than the intended relationship, which means that a rule branch that did not fire leaves no lineage record, and that it requires the execution platform to expose the necessary events.
- **Declarative specification** derives lineage from a mapping definition that is itself the executable artefact. Where the transformation rule is expressed declaratively, its lineage is a property of the specification rather than something that must be reverse-engineered. This is the most reliable mechanism and the least frequently available, because it requires the institution to have standardised on metadata-driven processing rather than permitting arbitrary code. The practical architecture combines the mechanisms: declarative specification where the processing framework supports it, static analysis for code-based transformations, and execution instrumentation as the reconciliation between intended and actual behaviour.

Attaching Context to Lineage

Lineage is necessary but not sufficient for traceability. A lineage edge that records that column A feeds column B answers little without the context of when, by what and under what conditions. Four contextual elements should be attached to every lineage edge. The transformation identity names the rule or process that performed the change. The version identity states which version of that transformation was in force. The execution identity states which run performed it. The effective period states the period for which the relationship held. With all four elements captured, the lineage graph becomes temporally queryable. The institution can ask not merely how a figure is produced now, but how it was produced in the period for which it was reported. Without the temporal element, the lineage graph describes the present and cannot answer the historical question, which is the question that examinations actually ask.

Lineage as a Release Gate

The most effective mechanism for keeping lineage current is to make its presence a condition of deployment. If a change to a transformation cannot be released unless it declares its input and output columns to a governed mapping, then the lineage graph is maintained as a by-product of the change process rather than as a separate exercise. Making lineage a release gate requires two supporting capabilities. The first is a place to declare the mapping, which may be a metadata file versioned with the code or a registry entry created through a workflow. The second is a validation step that checks the declared mapping against the actual structure, catching the case where a column is added to a query without being declared. Where organisations adopt this discipline, lineage coverage tends to be high because it is maintained by the process that changes the system. Where they do not, lineage is maintained by periodic projects, and its coverage decays from the moment each project ends.

Traceability in Operation

Traceability is a capability, and its value is realised in specific operational situations. Four are characteristic in regulatory data platforms.

- **Answering a supervisory question.** A supervisor asks how a particular reported figure was derived and what data underlies it. The response requires the lineage graph for the target cell, the rule versions in force for the period, the source attribute definitions, the reference data versions applied, and the control results for the period. Where the platform supports a query for each of these, the response takes hours. Where it does not, the response takes weeks and is assembled by engineers reading code.
- **Assessing the impact of a change.** A source system proposes to rename a column. The response requires the ability to identify every downstream asset that consumes it, every reporting obligation affected, and every historical period that might be restated. This is the reverse traversal of the lineage graph, and it is the single most valuable routine application of lineage in operational practice.
- **Reconciling a discrepancy.** Two reports that should agree do not. The response requires the ability to trace both figures back to their common source and identify the point at which they diverged. This is the application of lineage to the reconciliation process described in Chapter 9, and it converts reconciliation from an investigative exercise into a diagnostic one.
- **Reproducing a historical report.** A report must be reproduced for a past period, either for an examination or for a restatement. The response requires the ability to reconstruct the state of the pipeline as at that period: the data as extracted, the rules as they stood, the reference data as published, and the code as deployed. Reproducibility is discussed in Chapter 9 and is the most demanding traceability requirement.

Practical Limits of Traceability

It is worth being honest about the limits, because an architecture that promises universal traceability and delivers partial traceability damages credibility. Traceability can be incomplete where transformation logic is genuinely dynamic: where a target is computed from a configuration-supplied expression that is evaluated at runtime, static analysis cannot determine the relationship in advance. It can be incomplete where data passes through a manual step: a spreadsheet adjustment applied to a file leaves no automatic lineage record, though it should leave a workflow record. It can be incomplete where legacy code is undocumented and no longer understood by anyone currently employed.

The engineering response is not to claim completeness but to measure and report coverage. A lineage capability that reports ninety-two per cent column coverage and identifies the specific eight per cent that is not covered is more useful, and more defensible, than one that claims completeness without evidence.

Metadata Quality

Metadata is itself data and is subject to quality requirements. Three are particularly relevant.

- **Completeness** of the catalogue is measured as the proportion of data assets that are registered. Registered assets that have no owner, no description and no classification are registered but not curated, and a mature catalogue reports these separately.
- **Currency** is measured as the lag between a change in the underlying asset and the corresponding update in the catalogue. Where the catalogue is updated by crawling, currency is generally high for technical metadata and low

for business metadata. Where it is updated by the change process, currency depends on the process being enforced.

- **Accuracy** is measured by sampling: comparing the catalogue's description of a sample of assets against the assets themselves. Accuracy failures in metadata are more damaging than accuracy failures in the data, because metadata is used to form judgements about data.

Worked Example: Impact Assessment for a Proposed Source Change

A source system proposes to split a single customer identifier field into separate fields for the legal entity and the operating unit. The change is operationally motivated and the reporting function is not consulted at design time. What follows illustrates why impact assessment must be a query rather than an exercise in recollection. The lineage graph is traversed in reverse from the affected column. It identifies four staging attributes derived from it, two standardized-layer columns, three derived measures including one used in exposure aggregation, a reconciliation key used in two reconciliations, and eleven report data points across three returns. The reference data consumers are identified separately: two mapping tables resolve against the identifier's structure.

The assessment also reveals something the change proposal did not anticipate: the identifier is used as part of the natural key that makes ingestion idempotent, so a structural change to it affects duplicate detection as well as content. Without the lineage traversal, this consequence would have been discovered when duplicate records appeared in a reported total. The assessment's output is a change plan with a sequencing constraint, a testing obligation for each affected rule, and a statement of which historical periods cannot be reprocessed if the change is applied retrospectively.

Emerging Approaches to Lineage Capture

Research and practice have converged on three developments that reduce the cost of lineage capture.

The first is the open lineage model, a community standard for exchanging lineage metadata between platforms, which allows lineage captured by one tool to be consumed by another rather than remaining siloed. Its practical value is that an institution can combine lineage from an orchestration platform, a transformation engine and a database without building bespoke integrations. The second is query-level parsing with semantic resolution, in which a parser reads transformation logic and resolves the identifiers against a catalogue, so that the resulting edges reference governed assets rather than ambiguous names. Earlier parsers produced edges between unqualified names, which meant that two identically named columns in different schemas were merged. The third is execution-time capture at the plan level, in which a processing engine emits the resolved data flow of an execution rather than merely its inputs and outputs. This produces the why- and how-provenance described earlier at a granularity that static analysis cannot reach, at the cost of depending on the engine's cooperation.

Storing Lineage: Edge Models and Query Patterns

Lineage is frequently described conceptually and implemented casually, and the implementation choices determine which questions the capability can answer. Three storage models recur.

The node-and-edge store represents datasets, attributes, rules and reports as nodes and the dependencies between them as directed edges, each edge carrying a type such as derivation, filtering, aggregation or key resolution. Its strength is that path queries are natural: what feeds this report, what does this attribute feed, which rules resolve against this reference set. Its weakness is that the graph grows quickly and query performance depends on careful indexing by asset and by edge type. The run-scoped record captures lineage as an execution artefact: this execution consumed these inputs at these versions and produced these outputs. Its strength is evidential, because it records what actually happened rather than what ought to happen, and it is the only model that can distinguish a planned rerun from an unplanned one. Its weakness is volume, and the fact that it answers only questions about executions that were recorded.

The declarative specification derives lineage from the transformation definitions themselves, so that the edges are a computed consequence of the code. Its strength is that it stays current without manual maintenance, and its weakness is that it describes intent rather than behaviour, so that a transformation whose behaviour has diverged from its specification produces lineage that is confidently wrong. Mature implementations combine all three: the graph for path queries, the run

records for evidence, and the declarative derivation to keep the graph current. The combination is justified by the fact that each model's weakness is the next model's strength.

Column-Level Granularity and Its Cost

Column-level lineage is the granularity that regulatory frameworks require, because the question asked is almost always about an attribute rather than a table: which source column, under which rule, in which version, produced this reported value. Table-level lineage answers a different and easier question and is insufficient for that purpose. The cost of column-level capture is concentrated in three places. Transformation logic expressed in procedural code is difficult to analyse, so that column-level edges must be inferred rather than derived. Aggregations collapse many attributes into few, so that the reverse traversal from an aggregate to its contributing columns requires the aggregation's grouping to be recorded. And renames and casts are invisible in the output schema, so that an attribute whose meaning changed while its name did not produces an edge that is structurally correct and semantically misleading. Each of these is a reason to record the mapping explicitly at the point of transformation rather than to attempt to reconstruct it afterwards.

Lineage for Reference Data and for Rules

Lineage graphs are often built for data and left incomplete for the two asset classes that determine whether a reported figure is correct. Reference data lineage records which rules and mappings resolved against which version of a reference set, and when that version took effect. Without it, a change in a reported classification cannot be attributed to the reference change that caused it, and a historical period cannot be reconstructed because the set in force at that date is unidentified. Rule lineage records which derivation rules contributed to a reported figure, in which version, and under which parameters. This is the substance of what regulatory frameworks describe as attribute-level lineage, and its absence is the single most common reason that a past figure cannot be explained without reading source code.

Metadata Quality as Its Own Discipline

A metadata repository is itself a system, and it fails in the ways other systems fail: entries are created and never retired, definitions drift from the data they describe, ownership fields point at people who have changed roles, and lineages accumulate edges that no longer exist because a transformation was removed without updating the graph.

The discipline that prevents this treats metadata as a governed data domain with the same controls applied to it: completeness controls asserting that every reported data point has a mapped source, validity controls asserting that every named owner exists in the directory, currency controls flagging definitions not reviewed within their review period, and orphan controls identifying assets that no consumer references. Applying to metadata the controls that the platform applies to data is the clearest signal that an institution regards its metadata as an asset rather than as a record of an exercise.

CHAPTER 7

Data Ingestion, Integration, and Regulatory Data Pipelines

The preceding chapters established what data exists, what quality means and how meaning and provenance are recorded. This chapter addresses the movement of data into the platform. It treats ingestion as a controlled engineering process with defined completeness semantics, identified failure modes and prescribed recovery paths, rather than as a transport concern. The central claim of the chapter is that ingestion's hardest responsibility is not moving data but establishing that the movement was complete and that the result is attributable. Any competent engineer can write a program that copies rows from one system to another. Establishing that the copy contains every row that should be there, that it corresponds to the period claimed, and that it can be reproduced is a different problem, and it is the problem that regulatory reporting imposes.

Batch and Streaming Ingestion

Figure Batch and Streaming Ingestion Patterns compares the two principal ingestion modes. The comparison is presented as complementary rather than competitive, because most institutions operate both and the engineering task is to allocate workloads appropriately.

Scheduled batch extraction retrieves a defined set of data at a defined time. Its principal advantage in a regulatory context is that it produces a bounded population that can be validated as a set before release. Reconciliation, control-total comparison and completeness checking all operate naturally on a bounded population and are awkward on an unbounded stream. Batch also provides natural recovery semantics: a failed batch can be re-extracted from a known point without ambiguity about what has and has not been processed.

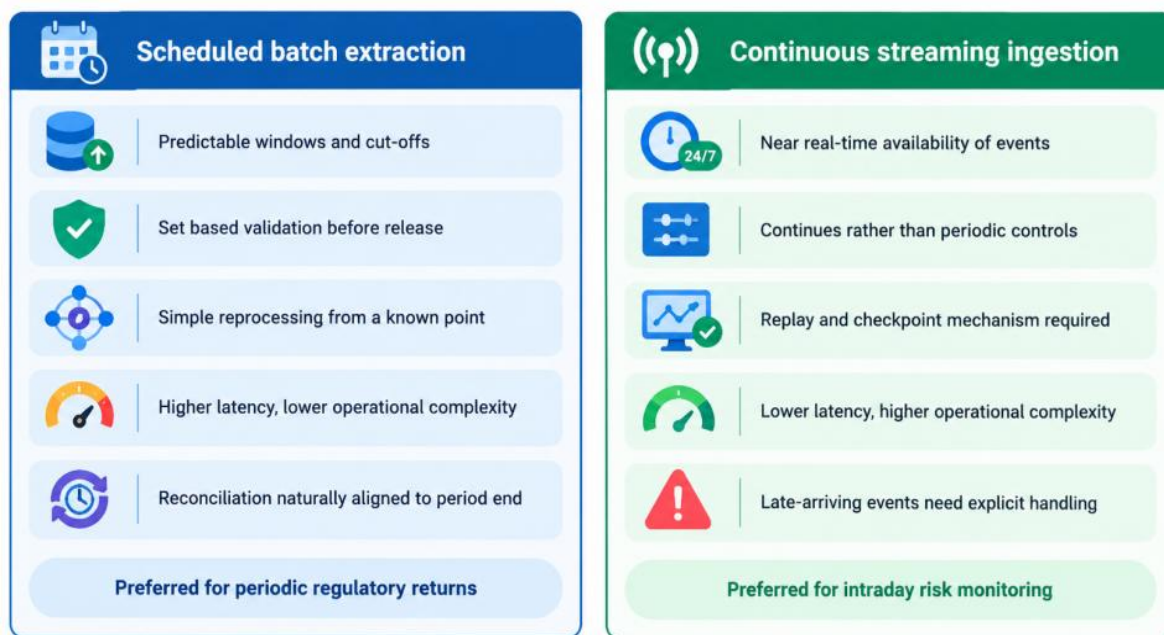


Figure 7.1 Batch and Streaming Ingestion Patterns

Continuous streaming ingestion delivers events as they occur. Its principal advantage is latency: a risk position that must be monitored intraday cannot wait for a nightly batch. Its principal difficulty in a regulatory context is completeness. An

unbounded stream has no natural denominator, so the completeness semantics that batch extraction obtains for free must be constructed explicitly from latency expectations, sequence continuity and reconciliation against a periodic reference.

The allocation decision follows from the obligation. Periodic regulatory returns are well served by batch extraction with a fixed cut-off, because the return is a statement about a period and the period boundary is the natural population boundary. Intraday risk aggregation, real-time exposure monitoring and event-driven conduct surveillance require streaming. Where an institution requires both, the characteristic pattern is to stream events for timeliness and to reconcile the stream against a batch extract periodically, so that the batch provides the denominator that the stream cannot.

Change Data Capture

Where the source is a relational database under the institution's control, change data capture retrieves only the records that have changed since the last extraction. Its advantages are reduced load on the source system, reduced volume to transfer and reduced latency. Its costs are complexity and a class of failure modes specific to it. The principal operational risk in change data capture is that the capture mechanism is separate from the application and can therefore fail in ways the application does not notice. A capture process that stops is not an error in the source application, and the application continues to operate normally. The defect accumulates silently until an extraction is compared against a control total and the gap is discovered. The control that addresses this is periodic reconciliation of the captured change volume against an independent measure of change in the source. Where the source maintains an audit trail or a modification timestamp that the institution can query independently, a periodic comparison of counts detects a stalled or partially failing capture.

Interface and File-Based Integration

Many sources cannot be queried directly, either because they are third-party systems or because direct access imposes unacceptable operational risk. In these cases integration proceeds through interfaces or through file exchange. Interface-based integration calls a defined application interface and receives structured data. Its advantage is that the interface contract is explicit and versioned by the provider. Its limitation is that the interface exposes what the provider chose to expose, which may be narrower than the reporting requirement. Where the gap is material, the institution must either negotiate an interface extension or obtain the data by another route. File-based integration receives a file delivered to a landing area. Its advantage is universality: any system can produce a file. Its difficulty is that the file's arrival is not evidence of its completeness, its content, or its correspondence to the period claimed. The controls that establish these properties are described in the next section.

The Ingestion Pipeline

Figure Regulatory Data Ingestion Pipeline presents the ingestion pipeline as five stages. The stages are presented in the order in which they should execute, and the ordering is a control decision.

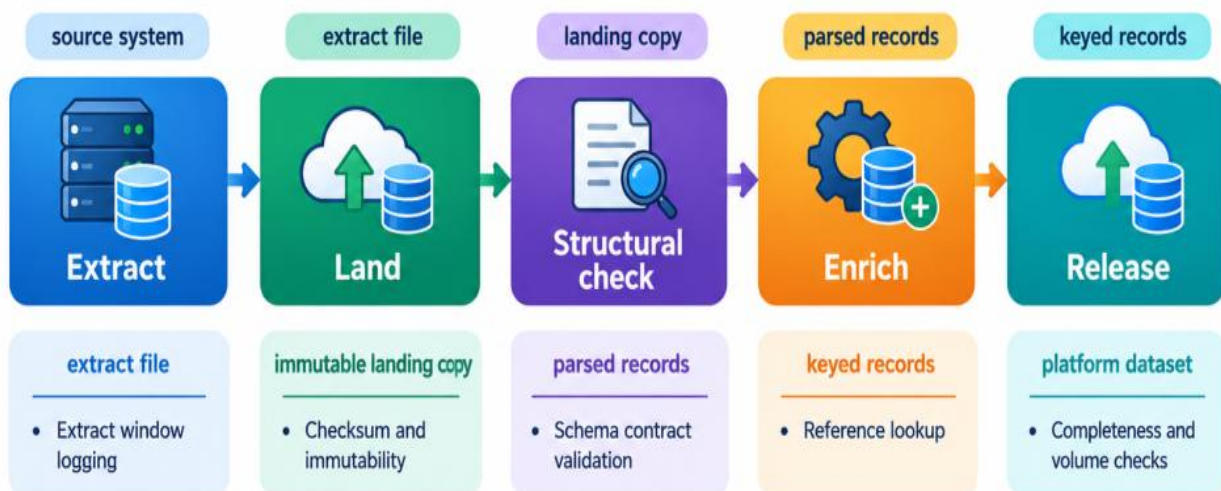


Figure 7.2 Regulatory Data Ingestion Pipeline

Extract

The extract stage retrieves data from the source. Three decisions are made here and should be documented, because each has downstream consequences. The extraction window determines the period covered. The decision is subtle because the natural answer extract everything created during the period is frequently wrong. Records may be created during the period but valued in the next, or created earlier and modified during the period. Where the reporting obligation is about a position at a date rather than about activity during a period, the correct extraction is of the position, not of the activity, and the two are different queries against the same system. The extraction completeness assertion states what the extract claims to contain. Where the source can supply a control total a count, a sum, a hash the assertion should be captured with the extract. Where it cannot, the extract should carry whatever partial assertion is obtainable. The extraction identity records which version of the source, which version of the extraction logic and which parameters produced the extract. This record is what makes the extract reproducible and is frequently omitted.

Land

The landing stage writes the extracted data to an immutable location in the platform. Immutability is the property that matters, and it is worth justifying because it imposes storage cost. If the landed copy is mutable, then a later correction to the source system can change the landed data, and the ability to reproduce a report produced from that data is lost. An immutable landing zone preserves the data as received. Corrections arrive as additional landed files, which the pipeline applies according to a defined rule, so that both the original state and the correction are retained and the sequence is reconstructible. Immutability also provides the recovery basis for reprocessing. Because every input is retained, a pipeline can be re-run for any historical period, producing either confirmation of the prior result or a controlled restatement.

Structural Validation

The structural validation stage applies the schema contract described in Chapter 2. It verifies that the expected columns are present, that their types are as declared, that the file structure is well formed and that the row count matches the control assertion where one exists. A schema failure should be a hard stop. Data whose structure does not conform cannot be interpreted reliably, and proceeding produces systematic misinterpretation rather than a detectable error. The recovery path is to reject the delivery, notify the source owner and re-extract. Schema validation is also the mechanism by which unannounced source changes are detected. A source that adds, removes or retypes a column will fail schema validation on the next delivery, which converts a silent defect into an explicit incident.

Enrichment and Keying

The enrichment stage attaches the identifiers required by downstream processing: the entity key from the master data store, the instrument key from the instrument reference service, the organisational unit key from the hierarchy. Keying is a control point as well as a transformation. Where an entity cannot be keyed, the record cannot participate in entity-level aggregation, and the aggregate will be incomplete.

The unkeyed records must therefore be routed to an exception queue and their volume must be controlled against a threshold. An institution that permits silent unkeyed records will produce entity-level aggregates whose completeness cannot be established. The enrichment stage is also where the reference data version used for keying should be recorded. Keying against a different reference data version than the one used for reporting produces inconsistency that is extremely difficult to diagnose after the fact.

Release

The release stage makes the data available to the platform. Its control responsibilities are volume validation and completeness assertion. Volume validation compares the delivered volume against an expected range. A range rather than a point is used because legitimate variation exists. The range should be derived from the historical distribution with a statistical basis rather than from a round number, and its breaches should be investigated rather than merely logged.

Completeness assertion is the strongest form of the control and the hardest to implement. It asserts that the delivered population is the complete population for the period. It requires an independent denominator, obtained from a source control total, an independent query, or a comparison against the prior period adjusted by an independently known

movement. Where the institution can obtain an independent denominator, it should assert completeness explicitly. Where it cannot, it should record that limitation, because a completeness claim that is not supported by an independent measure is not evidence.

Integration Topology

Figure Integration Topology for Heterogeneous Sources presents an integration topology for heterogeneous sources, with source adapters normalising into a canonical model from which consumer interfaces are served.

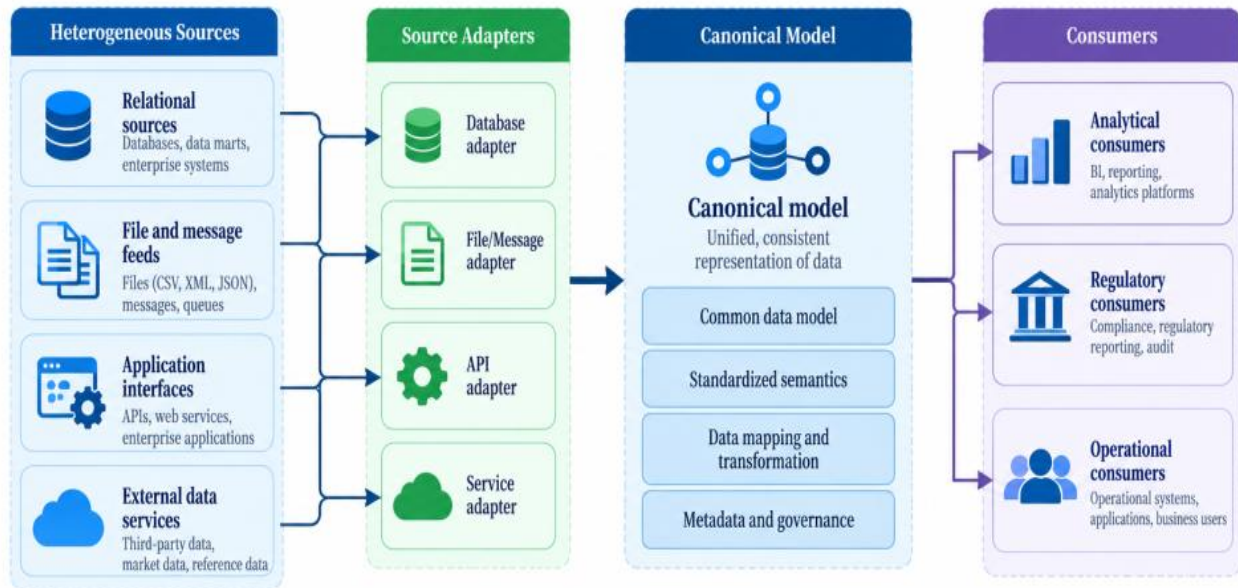


Figure 7.3 Integration Topology for Heterogeneous Sources

The topological choice that matters most is the location of the canonical model. Points-to-point integration connects each source directly to each consumer. It is fast to deliver for the first consumer and produces a combinatorial explosion of maintenance obligations thereafter: with four sources and four consumers, point-to-point integration potentially requires sixteen interfaces, each with its own mapping logic, its own controls and its own failure modes.

Introducing a canonical model converts the combinatorial problem into a linear one. Each source is mapped to the canonical representation once; each consumer is served from it once. The cost is the design and governance of the canonical model itself, which is a genuine and frequently underestimated expense. A canonical model that is poorly governed becomes a bottleneck in which every change requires central negotiation, and its cost can exceed the maintenance burden it was introduced to avoid. The practical criterion is the number of sources and consumers and the rate at which they change. With few of each, direct integration is often correct. With many, the canonical model is necessary, and the governance investment must be made deliberately rather than discovered.

Canonicity and Its Limits

A canonical model is a representation that is common to all sources and consumers. Achieving canonicity requires resolving semantic differences between sources, and the resolution is rarely perfect. The typical outcome is a canonical model that preserves the common case in a standard form and accommodates the divergent cases through extension attributes or through a residual representation that preserves the source form. The design decision is how much divergence to accommodate, and the trade-off is between the cost of resolving divergence at the boundary and the cost of carrying it through the model.

The engineering guidance is to canonicalise what is used and to preserve what is not. An attribute that is mapped into a common form and consumed by many downstream processes justifies the cost of standardisation. An attribute that is source-specific and rarely used should be carried through in its source form, marked as non-canonical, so that consumers

know not to rely on interoperability. Discarding it is worse, because the need for it will eventually arise and the source data may by then no longer be available.

Failure Modes and Controls

Figure Ingestion Failure Modes and Controls sets out the failure modes that a regulatory ingestion architecture must recognise, together with their detection and containment. Table 7.1 presents the same material in the form used in control documentation, with the addition of the recovery action and the accountability for each mode.

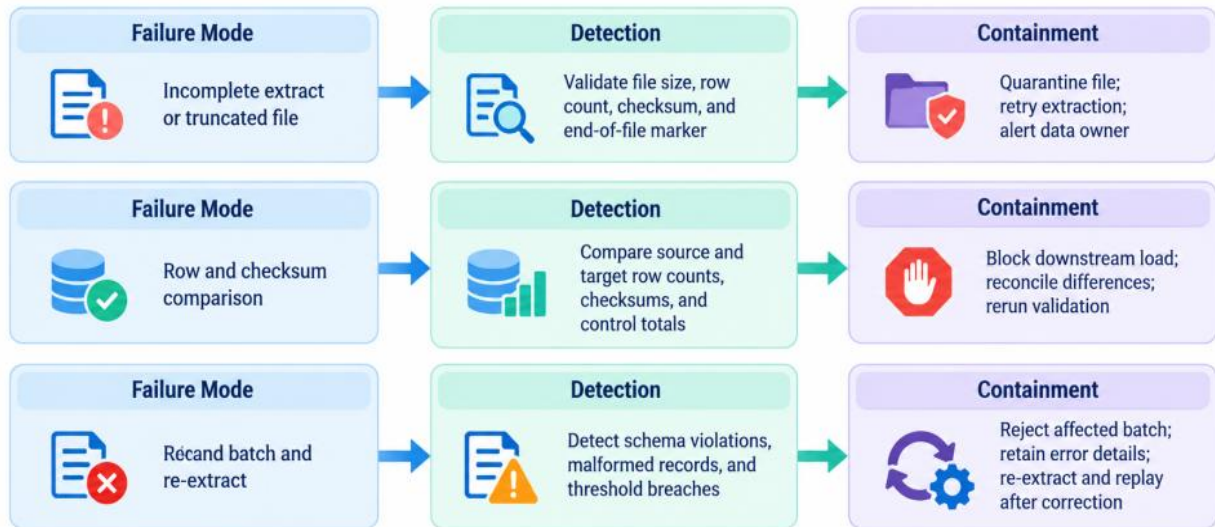


Figure 7.4 Ingestion Failure Modes and Controls

Table 7.1 Ingestion failure modes with detection, containment and recovery

Failure Mode	Detection Mechanism	Containment	Recovery
Truncated or partial file	Row-count and checksum comparison against control total	Reject delivery, do not release	Re-extract from source
Late delivery	Arrival monitoring against schedule and cut-off	Hold pipeline at the defined cut-off	Process on arrival with a recorded exception
Duplicate delivery	Natural-key comparison against landed history	Suppress duplicate, retain both deliveries	Determine cause of duplicate extraction
Schema change	Schema contract validation on delivery	Halt pipeline before enrichment	Agree interface revision with source owner
Unkeyable records	Key-resolution failure count against threshold	Route records to exception queue	Correct reference data or master data record
Volume anomaly	Comparison against historical distribution range	Hold for review before release	Investigate source or extraction defect
Silent capture failure	Periodic reconciliation of change volume to independent measure	Backfill from source if available	Repair capture, re-establish monitoring
Field-level semantic drift	Value-profile comparison against expected distribution	Raise exception on deviating attributes	Confirm change with source owner, revise rule

The table repays close reading because it illustrates a general principle of control design: detection, containment and recovery are three separate responsibilities that may be assigned to different mechanisms. Detection identifies the failure. Containment limits its consequence. Recovery restores the correct state. A control framework that detects without containing merely records the failure while it propagates; one that contains without recovering produces a permanent gap.

The Cut-Off Decision

No discussion of regulatory ingestion is complete without addressing the cut-off, because the cut-off is where engineering reality collides with legal obligation. The reporting obligation specifies a deadline. The pipeline requires a period during which to execute. The data arrives on a schedule determined by systems that were not designed with the reporting deadline in mind. The result is that the institution must choose a cut-off point after which late-arriving data is not included in the current return.

The choice is not arbitrary and should not be made silently. Three properties should characterise it. The cut-off should be documented and stable, so that it becomes a specification rather than a variable. It should be consistent with the period semantics of the sources, so that the cut-off does not truncate a source mid-period. And data excluded by the cut-off should be quantified and carried forward, so that its effect is visible in the following period rather than merely lost. An institution that applies its cut-off consistently and records what it excludes has a defensible position. An institution whose cut-off varies with operational convenience and whose exclusions are not quantified has a latent restatement risk that becomes visible only when a supervisor reconciles successive returns.

Idempotence and Replay

Ingestion processes should be idempotent: re-executing the same delivery should produce the same platform state rather than duplicating it. Idempotence is required for safe recovery, because without it every retry risks duplication. Achieving idempotence requires a natural key or a delivery identity by which duplicates can be recognised, and a defined behaviour on recognition. The behaviour may be to suppress the duplicate, to replace the prior content, or to raise an exception and require a decision. The choice depends on whether the source can legitimately deliver the same record twice with different content, which is common where records are amended in the source and re-extracted.

Where records are amended in the source, the platform must adopt an explicit treatment of versions. The two viable approaches are to treat each delivery as a new version and to select the appropriate version at consumption time, or to apply updates to the landed representation while retaining the prior version. The first approach is simpler and preserves history naturally; the second is simpler for consumers. The decision should be made once for the platform rather than separately by each pipeline, because consistency across pipelines is what makes the platform's temporal semantics comprehensible.

Worked Example: Controlling a File-Based Feed

File-based feeds illustrate the control burden that interface-based integration avoids. The following sequence describes the controls applied to a single daily file delivery.

The file arrives in a landing directory. Its arrival is detected and the arrival time is recorded. The file's name is parsed against a naming convention that encodes the source, the period and a sequence number, and a name that does not conform is rejected rather than guessed at. The checksum in the accompanying control file is verified. The row count in the control file is compared against the actual row count after parsing, and a mismatch halts processing. The period encoded in the file is compared against the expected period for the delivery date, and a mismatch raises an exception rather than being silently accepted. The file is copied to the immutable landing zone with its metadata, and the original is retained for the extraction window. Structural validation applies the schema contract.

Duplicate detection compares natural keys against the landed history, and a full duplicate is suppressed with a recorded event. Enrichment attaches keys, with unkeyed records routed to an exception queue and their proportion controlled against a threshold. Volume validation compares the delivered count against the historical distribution, and a deviation outside the range holds the batch for review. Completeness assertion compares against the source's control total where one

is supplied. Only then is the data released to the platform. Fourteen control points for one file, and each exists because its absence produces a defect that is expensive to diagnose after aggregation. The control set is the price of file-based integration, and it should be recognised as such when the integration approach is chosen.

The Canonical Model's Governance Requirements

Where a canonical model is introduced, four governance decisions must be made and recorded. The first is the admission process: what must be true for a new attribute to be added to the canonical model. An admission process that accepts every request produces a model that is canonical in name only. The second is the semantic owner of each canonical attribute, who is accountable for its definition and for deciding what happens when a source cannot supply it.

The third is the residual representation policy: how source-specific attributes that cannot be canonicalised are carried through. The policy should require that such attributes be explicitly marked as non-canonical so that consumers cannot mistakenly rely on interoperability. The fourth is the deprecation process, which determines how a canonical attribute ceases to be used. Canonical models accumulate attributes that no consumer reads, and without a deprecation process the model grows without bound.

Batch and Streaming: Choosing Deliberately

The choice between batch and streaming ingestion is frequently made by preference rather than by requirement, and the consequences of the choice are lasting because they determine the platform's control model, its cost profile and the skills it requires [23], [65].

Batch ingestion is characterised by a defined cut-off, a bounded population, an idempotent load and a control cycle that can complete before the data is released downstream. Its cost is proportional to the volume processed and its control model is well understood. Streaming ingestion is characterised by continuous arrival, unbounded populations, per-record processing and a control model that must operate on windows rather than on completed populations.

The requirement-based test is whether any reporting obligation the platform serves must be produced from data that is fresher than the batch interval. Where none must, batch ingestion is the lower-risk choice and streaming adds cost, complexity and new failure modes without regulatory benefit. Where one must, the requirement is usually confined to a small number of populations, and the appropriate response is to stream those populations while retaining batch processing for the remainder, rather than to convert the platform.

Reference Data During Ingestion

The point at which reference data is applied during ingestion determines both the cost of the join and the reproducibility of the result. Applying reference data at ingestion, sometimes described as enrichment at load, produces records that are self-describing and cheap to query. It also freezes the reference version into the record, which is desirable for reproducibility provided the version is recorded alongside the value. The failure mode is that a correction to the reference data requires reloading every record that consumed the superseded version.

Applying reference data at query time defers the join and keeps the reference data under a single point of control. It allows a correction to be applied to all consumers at once, and it makes the reported figures dependent on the reference data as it stands when the query runs rather than on the version in force at the reporting date, unless the query is parameterised by as-of date. The design that satisfies both requirements is to enrich at load while recording the reference version as an attribute of the enriched record, and to make every downstream query parameterised by the reporting date rather than by the current date. The enrichment then costs a join once, and reproducibility is preserved by the recorded version [67].

Idempotency and Its Dependence on Keys

An ingestion process is idempotent when executing it twice with the same input produces the same state as executing it once. Idempotency is what allows a failed load to be retried without investigation, and it is the property that makes reprocessing safe. Idempotency depends entirely on the natural key. Where the key is a surrogate identifier assigned by the source, a reloaded record carries the same key and replaces its predecessor cleanly. Where the key is a composite of business attributes, a correction to one of those attributes changes the key, so that the reloaded record does not replace its

predecessor but is inserted alongside it, and the population now contains both the superseded and the corrected version of the same business fact. This is one of the most common causes of double counting in reporting pipelines, and it is prevented by defining a stable business key that does not change when an attribute is corrected, or by maintaining a mapping from historical to current keys.

Integration Boundaries and Ownership

Every integration boundary is a place where a defect can enter and where accountability is contested. The design should therefore make each boundary explicit and owned. Three rules reduce boundary disputes. Each boundary has a named owner on both sides, recorded in the metadata repository rather than in a document. Each boundary has a defined contract, expressed as a schema with types, nullability, permitted domains and, where relevant, a control total. And each boundary has a defined validation obligation: a check that the received data conforms to the contract, executed by the receiving side, with the result retained. The receiving side validates because the receiving side bears the consequence of non-conformance, and the sending side is informed of the failure because the contract is mutual [17], [31].

An enterprise data ingestion architecture that collects information from heterogeneous operational sources, including relational databases, core banking applications, trading message streams, external regulatory files, and third-party data services. The Independent Ingestion Adapters layer provides change data capture, batch extraction, stream adaptation, file transfer, and API connectivity. These adapters enable different data delivery mechanisms to feed a common ingestion control boundary while preserving the distinctions between source formats and ingestion methods.

Within the Ingestion Control Boundary, the schema contract validator checks incoming structures against expected schemas, the completeness checker identifies missing data, the duplicate detector identifies repeated records, and the cut-off controller manages delivery timing. Accepted deliveries proceed to Immutable Raw Landing, which preserves ingested source data for downstream processing and traceability. Failed deliveries, schema failures, and incomplete batches are routed to the Landing and Recovery Zone, where failed-delivery quarantine and replay management support investigation and controlled reprocessing. Schema updates and rule changes feed back into the ingestion controls, while monitoring and alerting provide operational visibility into failures and delivery issues.

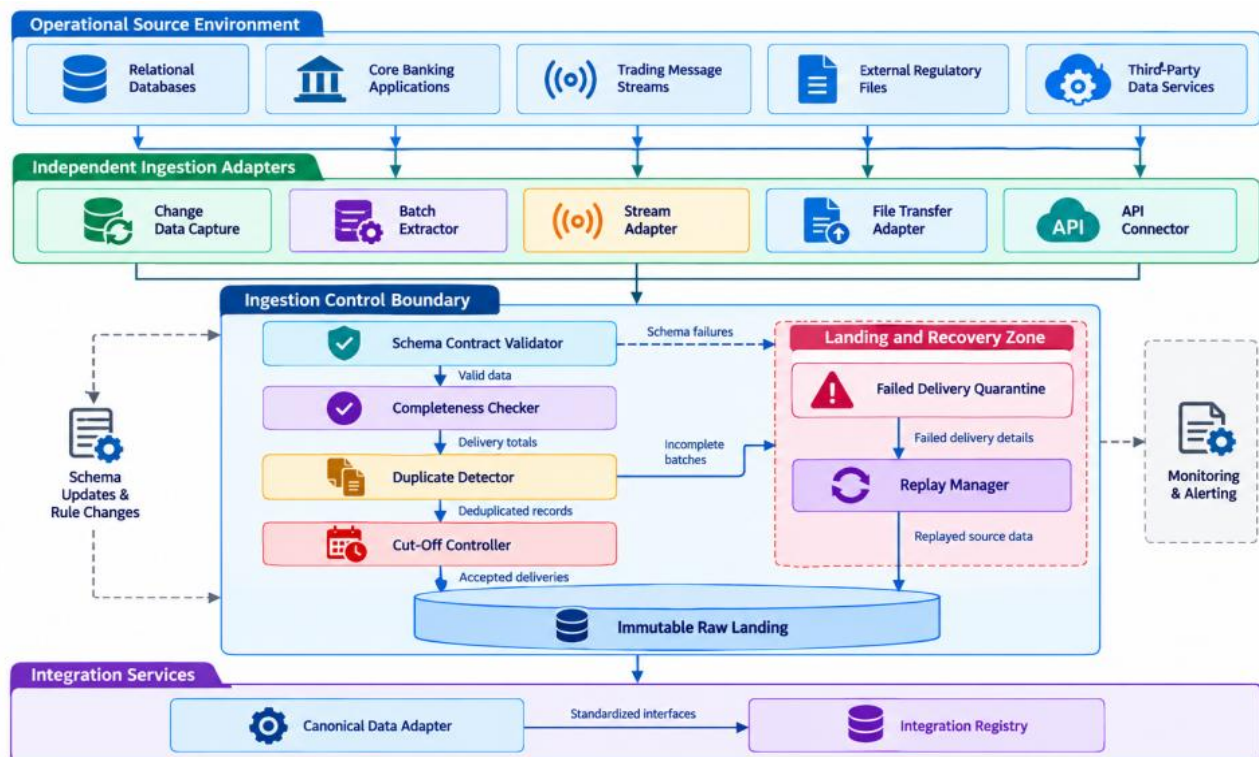


Figure 7.5 Enterprise Data Ingestion, Validation, and Resilient Landing Architecture

The Integration Services layer uses a canonical data adapter to expose standardized interfaces and an integration registry to maintain integration-related information. This separation between ingestion, control, recovery, and integration helps improve pipeline resilience and operational consistency. Overall, the architecture establishes a controlled ingestion process with validation gates, immutable raw-data retention, quarantine and replay capabilities, and monitoring support for reliable enterprise data processing.

CHAPTER 8

Data Transformation, Standardization, and Business Rule Execution

Once data has entered the platform, it must be transformed into the form the reporting obligation requires. This chapter treats transformation as the layer in which the institution's interpretation of its own data becomes explicit and executable. It is the layer where engineering and policy meet, because every transformation rule encodes a judgement about what a value means and how it should be treated. The central argument of the chapter is that transformation logic is not merely code. It is governed business policy expressed in executable form, and it must be managed with the same discipline applied to any other governed artefact: versioning, testing, approval, impact assessment and retention. An institution that treats transformation as implementation detail will discover during an examination that it cannot state, for a past period, which rules were applied to produce a reported figure.

The Nature of Transformation Work

Transformation performs four distinct kinds of work, and distinguishing them clarifies both the design and the governance of the layer. Figure Transformation Rule Taxonomy presents the four families.



Figure 8.1 Transformation Rule Taxonomy

Cleansing changes values that are present but defective. A null in a mandatory field is substituted with a default or a derived value. A free-text code that should have been an enumeration member is mapped to the nearest valid member. An out-of-range value is corrected, floored or flagged. Cleansing is the family that most directly affects reported figures, because a cleansing rule determines what value is used where the source value cannot be used directly.

Conforming changes representation without changing meaning. A currency amount recorded in minor units is converted to major units. A country expressed as a name is resolved to its coded representation. A classification expressed in an internal taxonomy is mapped to the regulatory taxonomy. Conforming rules are the mechanism by which the reference

data layer described in Chapter 5 exerts its influence over reporting, and they are the rules most affected by reference data change.

Deriving creates values that are not present in any source. A maturity bucket is derived from a contractual date and a reporting date. A ratio is computed from two measures. A risk weighting is selected from a lookup based on exposure class and rating. Derivation rules encode the analytical substance of the reporting framework and are the rules most likely to change when the framework changes.

Aggregating combines records into summaries. Aggregation is treated here as a rule family because the choice of aggregation logic which dimensions to group by, how to treat eliminations, which records to exclude is a rule with consequences, not a mechanical operation. Chapter 11 develops the aggregation architecture; this chapter addresses its rule semantics.

Why the Distinction Matters

The four families have different governance profiles, and conflating them produces both over-control and under-control. Cleansing rules require the strongest approval discipline, because they change values. A cleansing rule that substitutes a default is a statement that a missing value is to be treated as a specific value, and that statement may have material consequences for a reported total. Every cleansing rule should therefore be documented with its justification, its approval and its quantitative effect on the affected population.

Conforming rules require the strongest impact discipline, because they depend on reference data. When a reference set changes, every conforming rule that resolves against it is potentially affected. Identifying those rules requires the lineage capability described in Chapter 6, and the identification should be automatic rather than the product of recall. Deriving rules require the strongest testing discipline, because they encode policy that may be revised. A change to a derivation rule can change reported figures without any change in the underlying data, which means that the effect of the change cannot be assessed by examining the data alone. The practical control is a parallel-run comparison in which the revised rule is executed alongside the current rule and the differences are quantified and explained before the change is released.

Aggregating rules require the strongest reconciliation discipline, because an error in aggregation is an error in the reported figure itself and is invisible at the level of individual records. This is the subject of Chapter 9.

Standardisation as Executable Specification

Standardisation is the set of transformations that bring heterogeneous representations into a common form. It is presented separately because it is the layer at which the institution's semantic decisions become mechanical, and because it is frequently implemented as an ad hoc collection of string operations rather than as a governed specification.

Figure Business Rule Orchestration presents the orchestration of business rules as five stages. The stages are sequential in principle and interleaved in practice, but the ordering constraints they express are real and are frequently violated.

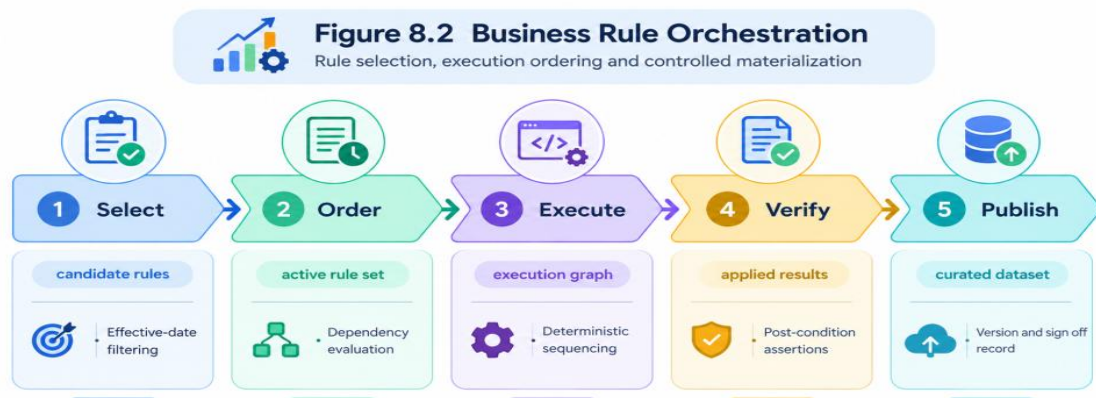


Figure 8.2 Business Rule Orchestration

Rule Selection and Effective Dating

The first stage selects the rules that apply to the data being processed. The selection is governed by effective dating: a rule applies to data whose reporting date falls within the rule's validity period. This is the mechanism by which a historical period can be reprocessed against the rules that were in force for that period. Effective dating is frequently absent, and its absence is one of the principal obstacles to reproducibility. Where rules are selected by version of the deployed code, the deployed code at any moment embodies only the current rules, and a historical re-run applies current rules to historical data. The result is a restated figure presented as a reproduction, which is a materially different thing and a serious control weakness.

The practical implementation stores each rule with an effective-from and effective-to date, retains superseded versions indefinitely, and selects rules by reporting date rather than by execution date. The practice imposes a discipline on rule authors: a rule change is a new rule with a new effective period, not an edit of the existing rule. That discipline is the same as the one applied to reference data in Chapter 5 and reflects the same underlying requirement.

Dependency Resolution and Execution Ordering

Rules depend on one another. A conforming rule may depend on a cleansing rule that has populated the attribute it maps. A derivation rule may depend on a conforming rule that has resolved a classification it uses. Executing rules in the wrong order produces either an error or, more dangerously, a silently different result. Two approaches to dependency resolution are used in practice. Explicit declaration requires each rule to state the attributes it reads and the attributes it writes, allowing the execution order to be computed from the dependency graph.

Implicit resolution infers dependencies from the expressions in the rules themselves. The first is more work for the rule author and is substantially more reliable, because it makes dependencies inspectable and allows cycles to be detected before execution rather than discovered as an unexpected result. Cycles are a real phenomenon, not a theoretical concern. A pair of rules in which each repairs a defect that the other can introduce is a common pattern, and it produces behaviour that depends on execution order. Where a cycle exists, the correct resolution is normally to restructure the rules so that the cycle is broken by an explicit sequence rather than left to depend on the order in which a scheduler happens to run them.

Determinism

Transformation must be deterministic: the same input, processed by the same rule set for the same period, must produce the same output. Non-determinism destroys reproducibility and makes reconciliation meaningless.

Three sources of non-determinism are common and should be actively excluded. The first is dependence on the current date or time, where a rule uses the system clock rather than the reporting date. The second is dependence on the order of unordered input, where an aggregation over a set without an explicit ordering produces results that vary with physical row order. The third is dependence on external state that changes independently, such as a reference lookup executed against the live reference service rather than against a versioned snapshot.

Each of these is avoidable, and each requires an explicit design decision. The reporting date must be a parameter rather than an ambient value. Aggregations must specify their ordering where the result depends upon it, or must be expressed as order-independent functions. Reference lookups must be resolved against a versioned snapshot whose identity is recorded with the output.

Idempotence and Incremental Processing

A transformation that can be executed repeatedly without changing its result is idempotent, and idempotence is a requirement for the same reason it is a requirement in ingestion: it makes recovery safe. An incremental transformation that adds to a target rather than replacing it is not idempotent, and re-executing it after a failure duplicates the increment.

Two architectural responses are used. The first is to partition the target by period and to rebuild each partition in full, which makes reprocessing trivially safe at the cost of recomputing more than necessary. The second is to make the increment explicitly keyed and replaceable, so that re-executing an increment for a period replaces the prior increment rather than adding to it.

The choice depends on volume and on the frequency of correction. Where corrections are frequent, full-partition rebuild is simpler to reason about and its additional cost is repaid in reduced investigation. Where the volume is large and corrections are rare, keyed incremental processing is appropriate, provided the keying is robust.

Business Rule Management

Figure Rule Versioning and Change Lifecycle presents the rule change lifecycle as a governed loop. The lifecycle applies to all four rule families and is the mechanism that keeps the rule inventory coherent.

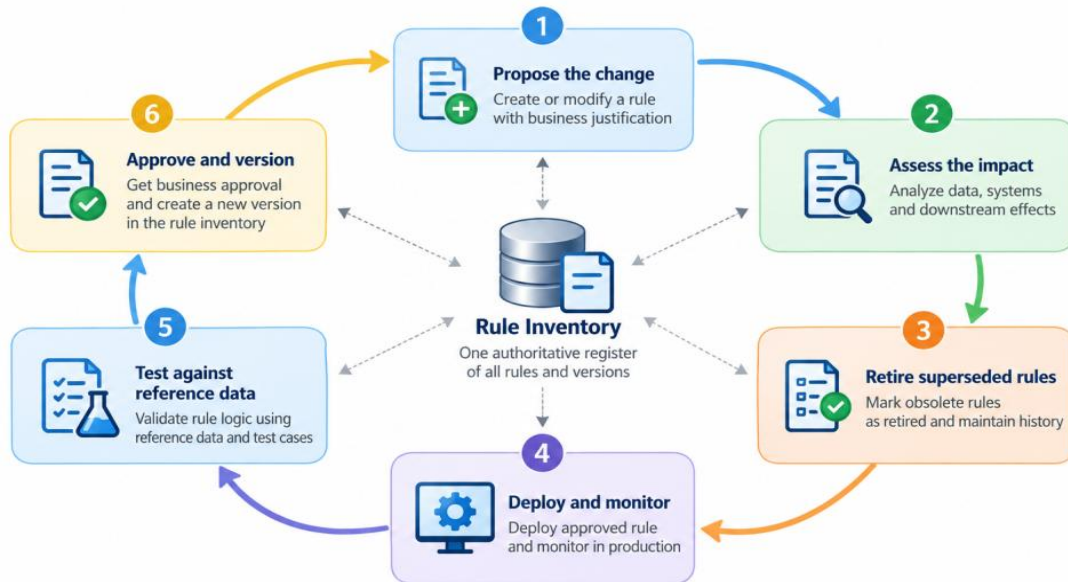


Figure 8.3 Rule Versioning and Change Lifecycle

The Rule Inventory

The foundation of rule management is a single inventory of every rule in force, with the metadata that allows each rule to be understood, tested and explained. Table 8.1 sets out the inventory attributes and their purposes.

Table 8.1 Transformation Rule Categories with their Validation Requirements and Governance Controls

Rule Category	Purpose	Validation Required	Governance Control
Cleansing	Repair or substitute defective values	Null-rate and substitution-volume monitoring	Documented justification and quantified effect
Conforming	Align representation with governed classifications	Resolution-rate monitoring against reference completeness	Impact assessment on reference data change
Derivation	Compute values not present in source data	Parallel-run comparison on change, boundary-value testing	Policy owner approval and effective dating
Aggregation	Combine records into reported summaries	Aggregate-to-detail reconciliation and control-total agreement	Reconciliation sign-off per period

Rule Category	Purpose	Validation Required	Governance Control
Temporal	Assign and validate effective periods	As-of consistency and sequence integrity checks	Period semantics confirmation with source owner
Integrity	Assert relationships between records	Referential, cardinality and cross-field arithmetic checks	Exception review and threshold ownership
Enrichment	Attach keys, hierarchies and classifications	Key-resolution rate and unresolved-record volume	Reference and master data stewardship
Selection	Determine which records enter the reported population	Population-count monitoring against independent expectation	Inclusion and exclusion policy approval

The inventory serves four functions. It provides the basis for impact assessment, since a proposed reference data change can be assessed against the conforming rules that resolve against the affected set. It provides the basis for testing, since the test obligation associated with a rule follows from its category. It provides the basis for the audit explanation of a past figure, since the rules in force for a period are recorded. And it provides the basis for rationalisation, since an inventory reveals duplicates, contradictions and obsolete rules that are invisible when rules are embedded in code.

Rule Testing

Testing obligations differ by rule category, and specifying them by category prevents both under-testing and the use of a single, heavy test process for every rule. Cleansing rules should be tested against a population containing the defect they address, with the test asserting the expected substitution and its volume. A cleansing rule tested only against clean data proves nothing.

Conforming rules should be tested against the full reference set in both directions: that every value in the source resolves, and that every resolution is correct. The first direction detects incomplete mappings, which are the most common defect. The second detects incorrect mappings, which are the most damaging. Derivation rules should be tested at boundary values, because derivations commonly involve thresholds and the errors cluster at the boundaries. A maturity bucket rule with a one-year boundary should be tested at the day before, the day of and the day after the boundary. Where the rule has been revised, a parallel-run comparison against the prior version is the strongest available evidence of intended effect. Aggregation rules should be tested by construction of a known population with a known correct total, which is the only way to establish that the aggregation logic is correct independently of the data it processes.

Rule Retirement

Rules are added in response to incidents, requirements and framework changes, and are rarely removed. The accumulation has two costs. The first is operational: every rule consumes execution time and generates results that must be reviewed. The second is epistemic and more serious: a rule set containing rules that are known to fire routinely on acceptable data trains its operators to disregard its output, which degrades the signal in the rules that matter. The remedy is a periodic review in which each rule is classified as current, superseded or redundant, with the classification recorded and removal approved by the rule's owner. The review interval should be tied to the framework's change cycle, and the review should be treated as a control in its own right rather than as housekeeping.

The Metadata-Driven Transformation Architecture

The design that makes rule governance tractable is one in which the rules are data rather than code. In a metadata-driven architecture, the transformation logic is expressed as a specification stored in a governed repository, and a generic

execution engine interprets the specification. The architecture has four components. The rule repository stores the specifications with their metadata and versions. The execution engine interprets specifications and applies them to data. The lineage capture records the rules applied to each output, deriving the record from the specification itself. The test harness executes rules against reference populations and asserts expected results.

The advantages are substantial. Lineage is obtained without reverse-engineering because the rule specification states its inputs and outputs. Impact assessment is a query against the repository. Historical reproduction is achieved by selecting the rule versions in force for the period. Testing is automated because the rule is a structured artefact. The costs are equally real. The expressiveness of the specification language limits what can be expressed; genuinely novel logic requires an escape hatch, and every escape hatch weakens governance. The engine is itself software requiring maintenance. Performance may be lower than hand-optimised code for high volumes, though modern engines have narrowed this gap substantially. The architectural judgement is that metadata-driven processing is worth its cost where the rules are numerous, change frequently and require explanation. Where a pipeline has few rules, changes rarely and is not subject to examination, hand-written code is a defensible choice. Since regulatory reporting is characterised by numerous rules that must be explained, the metadata-driven approach is generally the better fit, and the escape hatches should be governed explicitly rather than permitted implicitly.

The Escape Hatch Problem

Every metadata-driven architecture encounters logic that its specification language cannot express: a complex interpolation, a proprietary calculation, an optimisation for a volume that the generic engine cannot handle. The response determines whether the architecture's guarantees survive. An ungoverned escape hatch arbitrary code embedded in a pipeline reintroduces every problem the architecture was designed to solve. Its lineage is not captured, its version is not managed, its effect is not inventoried. A governed escape hatch treats the exception as a first-class artefact: it is registered, it declares its inputs and outputs so that lineage remains complete, it is versioned and tested by the same process as declarative rules, and it is reviewed to determine whether the specification language should be extended. The last element is important because it converts repeated exceptions into a language evolution rather than a permanent accumulation of special cases.

Worked Example: A Maturity Bucket Derivation

A derivation rule assigns each instrument to a maturity bucket defined by the number of days between the reporting date and the contractual maturity date. The rule appears trivial and it exercises most of the principles in this chapter. The rule must take the reporting date as a parameter rather than reading the current date, or a re-run for a historical period will produce different buckets. It must state its boundary treatment explicitly: whether an instrument maturing on the boundary day falls in the lower or upper bucket, and the treatment must match the reporting framework's specification. It must handle the missing case: an instrument with no maturity date, which occurs for perpetual instruments, requires a defined treatment rather than a null bucket. Its effective dating matters because the bucket boundaries are specified by the reporting framework and have changed between framework versions. A historical period must be processed with the boundaries in force for that period.

Its testing obligation includes boundary values: the day before, the day of and the day after each boundary. Boundary errors in bucket derivations are among the most common reported restatements, and they are detectable by a test that takes minutes to write. Its impact analysis matters because maturity bucketing frequently appears in more than one obligation, and a change to the boundaries affects every consumer. Where the rule is duplicated across pipelines rather than shared, the change must be applied consistently, which is an argument for the single-rule-inventory model.

A Rule Register Entry

Table 8.1's attributes are most easily understood as fields of a register entry. A conforming rule that maps an internal product code to the regulatory product classification carries: a rule identifier, a descriptive name, the rule category, the accountable owner, the effective-from date, the effective-to date, the input attributes with their source, the output attribute, the mapping table version it resolves against, the population it applies to, the completeness control that asserts every

source value resolves, the test cases with their expected results, the last review date, and the status of current, superseded or redundant. The register entry is what allows each of the operational situations described in this book to be answered as a query: which rules resolve against this reference table, which rules were in force for this period, which rules have not been tested, which rules no pipeline calls.

Cleansing Versus Correction

The distinction between cleansing and correction is frequently blurred and has practical consequences for accountability. Cleansing is a deterministic, reproducible transformation applied uniformly to a population: trimming whitespace, normalising case, resolving a recognised abbreviation to its standard form, converting a date to a canonical representation. It changes the representation without changing the information, and it is appropriate for the platform to perform it silently, provided the original value is retained. Correction changes the information: a misspelled name is set to its correct spelling, an obviously transposed digit in an amount is replaced by an amount derived from another source. A correction requires a basis, and the basis must be recorded. Where the platform corrects silently, the institution has no record that the source value differed, and a subsequent examination comparing the platform's value against the source's value will find an unexplained discrepancy [6]. The design principle is therefore that cleansing is performed in the pipeline and logged at the attribute level, while correction is performed in a governed remediation process in which each correction carries an identifier, an approving actor, a stated basis and an effective date. The correction is then itself an auditable event rather than an unexplained difference.

Ordering Rules and the Errors It Prevents

Where transformation involves several dependent steps, the order of execution is part of the rule's definition rather than an implementation detail. Five ordering constraints recur. Standardisation must precede matching, because matching against unstandardised values produces similarity scores that reflect the format rather than the entity. Reference resolution must precede derivation, because a derivation that consumes a classification requires the classification to be resolved. Filtering must precede aggregation unless the filter is applied deliberately after aggregation, because a control total computed before a filter cannot be reconciled against one computed after it. Currency conversion must follow the determination of the amount's currency, and must be applied once rather than at two stages, because applying conversion twice produces a figure that is wrong by the ratio of the two rates. And the as-of date must be resolved before any rule that depends on effective dating, so that the correct reference version is selected [19], [21]. A rule whose ordering constraint is violated does not fail visibly. It produces a plausible figure that disagrees with the correctly ordered figure, and the disagreement surfaces at reconciliation or, worse, at examination.

Determinism and Its Enemies

Determinism is the requirement that repeated execution of the same inputs produce the same outputs. It is treated as an obvious property of a data pipeline and it is routinely violated. Four causes of non-determinism recur. Nondeterministic functions, of which the current date and time are the most common, embed the moment of execution into the output. Unordered processing, where a rule consumes a set without a deterministic ordering, produces different results depending on the order in which the engine happens to process the records, which is a particular hazard for rules that select the first or last record within a group. Floating-point aggregation performed in parallel is not associative, so that summing a large population in a different partition order produces a slightly different total, which is tolerable for analytics and intolerable for a reconciliation that must agree exactly. And reference data resolved at execution time against the current version rather than the version in force at the reporting date changes the result whenever the reference data changes. Each cause is removable: the reporting date is passed as a parameter, the ordering is made explicit in the rule, the aggregation is performed in a defined order or reconciled to the ledger's total, and the reference version is selected by as-of date. Together these practices are what allow a reconciliation performed once to remain valid on re-execution, which is the practical meaning of reproducibility [28].

Testing Transformation Rules

The testing obligation for a transformation rule follows from its category, and a test strategy that treats all rules alike over-tests the trivial and under-tests the consequential. Mapping rules are tested by completeness: every value in the source

domain must resolve, and the test asserts that no value falls through to an unresolved default. Derivation rules are tested at their boundaries, as the worked example above illustrates. Aggregation rules are tested by reconciliation against an independent total. Standardisation rules are tested by round-tripping: a value that has been standardised and then expressed in its original form should be recognisable as an equivalent, and a value that the standardisation does not understand should pass through unchanged rather than being silently altered. Filtering rules are tested by population counts on both sides of the filter.

The most valuable test artefact is a set of records that are deliberately defective in ways the rule is intended to detect, retained with the rule and executed on every change. Constructing that set is the work that is habitually skipped, and its absence is why rule changes are frequently found to have broken a control months after deployment. The figure illustrates an enterprise architecture for transforming source data into a standardized regulatory dataset through governed business rules, controlled execution, and assurance mechanisms. The Transformation Inputs include landed source data, reference data versions, and reporting date context. These inputs feed the Rule Execution Environment, where standardization and service operations normalize data and resolve dependencies before the Rule Execution Engine applies the required rules and transformations. The resulting transformed information is delivered to the standardized regulatory dataset.

The Governed Rule Repository manages cleansing rules, conforming rules, derivation rules, and aggregation rules. These rule definitions and their versions are maintained in the Rule Version Registry, supporting controlled changes and reproducibility. The Assurance and Lineage layer provides a rule test harness, parallel run comparator, and lineage capture. These components support rule testing, comparison of execution outputs, and collection of evidence about how the transformed data was produced. The resulting rule-test information, calculated outputs, and lineage metadata help establish traceability between source inputs, rule versions, and regulatory outputs.

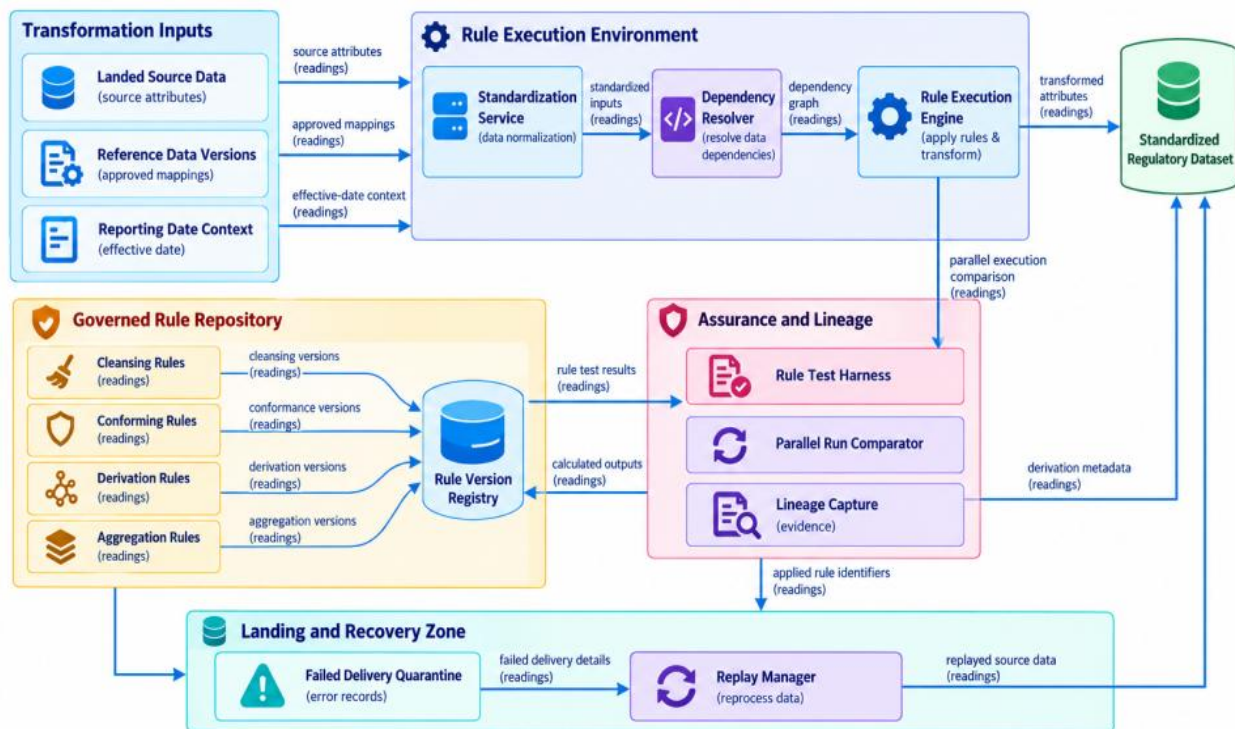


Figure 8.3 Governed Data Transformation, Rule Execution, and Lineage Assurance Architecture

The Landing and Recovery Zone handles failed deliveries through quarantine and replay management. Failed records can be investigated and reprocessed, with replayed source data returned to the transformation flow. By combining versioned rules, dependency-aware execution, lineage capture, testing, and controlled recovery, the architecture supports consistent regulatory data transformation and stronger operational assurance.

CHAPTER 9

Data Reconciliation, Completeness, and Financial Control Frameworks

Reconciliation is the control that establishes whether the aggregates a reporting system produces are correct. It is treated here as an architectural capability rather than a compliance activity, because its effectiveness depends on design decisions taken long before any reconciliation is performed. A reporting platform that has not been designed to support reconciliation cannot be reconciled by effort.

The chapter advances a specific position. Validation, as described in Chapter 4, asserts expectations about records. Reconciliation asserts relationships between aggregates produced by independent paths. The two are complementary and neither substitutes for the other, because validation can only detect violations of expectations that the institution has already articulated, while reconciliation can detect the consequence of an expectation the institution has not yet articulated.

Consider a pipeline that omits an entire product type from an aggregation because a filter condition was written more narrowly than intended. Every record that passes through the pipeline is valid. Every per-record rule succeeds. The completeness rule, if its denominator is derived from the same pipeline, also succeeds. The only control that detects the omission is a reconciliation against a total produced by a different path — a control total from the general ledger, or a prior-period figure adjusted by an independently known movement.

The Reconciliation Control Framework

Figure Reconciliation Control Framework presents the reconciliation framework as five stages. The framework is presented as a repeatable process rather than a one-off exercise, because reconciliation value comes from its regularity.



Figure 9.1 Reconciliation Control Framework

Stage One: Define the Basis

Reconciliation compares two numbers. Before it can do so, the institution must specify precisely what the two numbers represent and why they should agree. Specifying the basis is the most intellectually demanding part of reconciliation and the part most frequently skimmed. Two figures that should agree frequently do not agree exactly for legitimate reasons. An exposure figure computed from positions at the close of business may differ from the same exposure computed from the ledger because the ledger reflects a subsequent adjustment. The difference is not an error but it must be anticipated and quantified, because otherwise every period produces an unexplained variance that trains its reviewers to accept variances.

The specification should state the reporting line, the two sources, the expected relationship equality, or equality within a defined tolerance with a stated structural difference and the accountable owner who accepts that specification. The specification is itself an artefact requiring version control, because the relationship between two systems changes as the systems change.

Stage Two: Extract Control Totals Independently

The second stage obtains the two numbers. The requirement that matters is independence: the control total must be obtained by a path that is not the path being tested.

Independence has degrees, and the institution should be explicit about which degree it has achieved. At the strongest degree, the control total is obtained from a system of record by a query written by a team independent of the pipeline team, executed against source data rather than against platform data. At a middle degree, the control total is obtained from the source by a separate extraction using different logic. At the weakest degree, the control total is derived from the platform itself by an alternative query, which detects only those errors that the alternative query happens to avoid.

The weakest degree is common because it is cheap, and its limitation should be understood. It detects logic errors between two views of the same platform data but is blind to omissions and duplications that affect both views equally, since both read the same landed data.

Stage Three: Compare with Defined Tolerance

The third stage computes the difference and evaluates it against the tolerance specified in stage one. The tolerance should be expressed in the unit in which the risk is borne, as discussed in Chapter 4, and it should distinguish between a structural difference, which is expected and has a documented cause, and an unexplained variance, which does not.

The distinction between structural and unexplained variance should be enforced by the reconciliation tooling rather than by the reviewer. Where a structural difference is known and quantified, it should be encoded as an expected adjustment in the reconciliation definition, so that the residual presented to the reviewer is the genuine unexplained amount. Where this discipline is not applied, reviewers spend their effort re-deriving known differences each period, and their attention is not available for the differences that matter.

Stage Four: Investigate and Explain

The fourth stage assigns a cause to each unexplained variance. The investigation is a diagnostic exercise, and its efficiency depends entirely on the lineage and traceability capabilities described in Chapter 6.

Where column-level lineage is available, the investigation proceeds by identifying the point at which the two paths diverge: the common ancestor of the two figures, and the transformation at which they separate. Where lineage is not available, the investigation proceeds by elimination, which is slower and less reliable.

Every investigation should terminate in one of four conclusions. The variance is caused by a data defect, in which case remediation is at the data. It is caused by a rule defect, in which case remediation is at the rule and the rule change follows the lifecycle in Chapter 8. It is caused by a specification defect, in which case the reconciliation basis itself was wrong and must be corrected. Or it is caused by a timing difference, in which case it should be expected, quantified and carried forward. The four conclusions are worth separating because they imply different owners and different remediation. An institution that records only that a variance was explained loses the information required to prevent its recurrence, and analysis of the recorded causes is the most reliable source of evidence about where the platform's weaknesses actually lie.

Stage Five: Certify and Archive

The final stage records the signed reconciliation statement for the period and retains it. The statement should identify the reconciliation, the two values, the difference, the tolerance, the conclusion, the responsible person and the date. Retention must extend for the period required by the institution's records policy and by the applicable supervisory requirement.

The archived statement is the evidence that the control operated. In the absence of an archived statement, an institution may be able to demonstrate that a reconciliation process exists, but not that it produced a result for the period in question. The distinction is precisely the distinction drawn in Chapter 2 between a control and evidence of control operation.

Reconciliation Methods

Figure Reconciliation Methods and Matching Logic positions the principal reconciliation methods against the matching key and residual treatment appropriate to each. Table 9.1 presents the same methods in the form used in control documentation, adding the tolerance basis and the exception class each method generates.



Figure 9.2 Reconciliation Methods and Matching Logic

Table 9.1 Reconciliation Methods with Matching Key, Tolerance Basis and Generated Exception Class

Methods	Matching Key	Tolerance Basis	Exception class Generated
Aggregate-to-detail	Entity, product and period	Exact agreement against control total	Completeness exception
Record-to-record	Business key plus value	Exact match on all compared attributes	Unmatched or disputed record
Position-to-transaction	Instrument and value date	Reconstruction of position from movements	Timing or missing movement exception
Hierarchy roll-up	Parent and child entity keys	Arithmetic agreement plus elimination check	Elimination or hierarchy exception
Cross-system corroboration	Independent attribute set agreed between systems	Agreement within attribute-specific tolerance	Data disagreement exception

Methods	Matching Key	Tolerance Basis	Exception class Generated
External benchmark	Published aggregate or index	Directional and order-of-magnitude consistency	Benchmark deviation exception
Opening-to-closing bridge	Movement reconstruction across periods	Exact reconciliation of the movement set	Unresolved movement exception
Statistical plausibility	Distributional position of the value	Deviation from expected distribution	Anomaly exception requiring review

Aggregate-to-Detail Reconciliation

The most important method is the comparison of an aggregate against the sum of its constituents. It answers the question that supervisors ask most frequently: does the reported total agree with the underlying records? The method's effectiveness depends on the independence of the two paths. Where the aggregate is computed by summing the detail records that the platform holds, and the control total is likewise computed by summing the same records, the reconciliation is vacuous. The control total must come from a system that computes it independently: the general ledger's trial balance, the source system's own reporting, or an independently written query.

Record-to-Record Reconciliation

Record-to-record reconciliation establishes that two systems hold the same set of records describing the same events. It produces three outcomes: records present in both and agreeing, records present in both and disagreeing, and records present in only one. Records present in only one system are the most informative outcome and the most commonly underspecified. The comparison should distinguish, for each system, records that are absent from the other, because the direction of the difference carries information. A systematic excess in one system suggests a definitional difference. A systematic deficit suggests a pipeline omission.

Disagreements between records present in both require attribute-level comparison with a stated precedence. Where two systems disagree about an attribute, one of them is the system of record for that attribute by the domain decomposition described in Chapter 2, and the disagreement is a defect in the other. Where no precedence has been assigned, the disagreement cannot be resolved and the ambiguity should be treated as a governance gap rather than as a data defect.

Position-to-Transaction Reconciliation

For balance-sheet reporting, the strongest available control is the reconstruction of a position from its movements. The opening position, plus movements during the period, adjusted for the effects of revaluation and reclassification, should equal the closing position. The method is powerful because it constrains the data from two directions simultaneously. It tests the completeness of the movement set, the correctness of the movement classification and the arithmetic of the position calculation. An omission of a movement class will fail the reconstruction even if every individual movement record is valid. The method requires a movement taxonomy with exhaustive coverage, since the reconstruction must account for every change in the position. The taxonomy is the specification, and its completeness is the precondition for the method's effectiveness.

Hierarchy and Elimination Reconciliation

Where reporting requires consolidated figures, reconciliation must verify both the internal arithmetic of the hierarchy and the correctness of the eliminations. The internal arithmetic check verifies that each parent figure equals the sum of its children plus any directly held amount. The elimination check verifies that intra-group transactions and balances have been identified and removed. The second check is the harder one, because an omission in elimination produces a figure that is internally consistent at every level and overstated in aggregate.

The practical control is a bilateral check: for every intra-group relationship, the amount recorded by one party should equal the amount recorded by the counterparty. Where the two differ, the elimination is incomplete by the difference. The check requires both sides of the relationship to be identified, which requires the legal entity hierarchy described in Chapter 2 to be accurate and current for the reporting date.

Completeness as a Distinct Control Problem

Completeness deserves separate treatment because it is the quality dimension most directly connected to reconciliation and the one most frequently mis-controlled. Completeness is asserted at the level of a population. The population is defined by the reporting obligation and by the period. The control asserts that the platform holds every member of the population. There are three broad approaches, and institutions should be able to state which they use.

- **Control-total reconciliation** obtains a count or a total from an independent system and compares it against the platform's count or total. It is the strongest approach where an independent system exists.
- **Population reconstruction** derives the expected population from an independent basis: for example, deriving the expected set of accounts from the general ledger's chart of accounts rather than from the pipeline that reads accounts. It is intermediate in strength and commonly available.
- **Trailing-range monitoring** compares the current population size against the recent distribution and flags deviations. It is the weakest approach because it detects only changes, not persistent omissions. Its weaknesses should be understood: a population that has been five per cent incomplete for two years will produce no signal.

The practical design combines the approaches. Trailing-range monitoring is applied universally because it is cheap. Population reconstruction is applied to populations with a derivable expectation. Control-total reconciliation is applied to the material populations.

The Denominator Problem in Practice

The denominator problem stated in Chapter 4 reappears here in a specific form that is worth making concrete. Suppose the institution wishes to assert that all trades have been captured for a period. The natural denominator is the count of trades in the trading system. But this denominator is produced by the trading system, and a pipeline that reads the trading system's extract will obtain the same count as the trading system's own report. If the extract is incomplete, both the numerator and the denominator are equally incomplete and the control reports success. The resolution requires a denominator from outside the extraction path. Options include a count obtained by a direct query against the trading database executed by a separate process, a count from the counterparty confirmation process, or a count derived from the settlement system, which is downstream of the trading system but reached by a different path. Each is imperfect; each is substantially better than a denominator derived from the path being tested. Where no independent denominator can be obtained, the honest position is that completeness cannot be asserted, and the limitation should be recorded rather than papered over with a weaker control presented as adequate.

Financial Control Analogies and Their Limits

Regulatory data control has borrowed extensively from financial control, and the borrowing is largely beneficial. It is worth being precise about what transfers and what does not. Financial control, as articulated in the internal control frameworks discussed in Chapter 2, rests on several well-established techniques: segregation of duties, independent verification, reconciliation of independently maintained records, and documented authorisation. Each of these has a direct analogue in data control. Segregation of duties translates into the separation between those who build a pipeline and those who verify its output, and the separation between those who define a rule and those who approve it. Where the same team builds the pipeline, defines the rules and certifies the result, the control environment is deficient for the same reason that a single bookkeeper who reconciles their own ledger is deficient. Independent verification translates into the independent extraction path discussed above. Reconciliation of independently maintained records translates into cross-system corroboration. Documented authorisation translates into the approval workflow for rule changes and releases. What does not transfer cleanly is the notion of a materially exact reconciliation. Financial reconciliation commonly aims at exact agreement, because the two records describe the same monetary events. Data reconciliation frequently compares quantities that legitimately differ by timing, by definition and by treatment. The control objective for such comparisons is not exact agreement but agreement after the anticipated differences have been removed, and the discipline lies in specifying and quantifying the anticipated differences.

A second area where the analogy strains is the treatment of tolerance. Financial control tolerances are typically set at a level where the residual is immaterial. Data control tolerances are frequently set at a level where the residual is tolerable operationally. Where the two are conflated, the operational tolerance becomes the accepted materiality threshold, which is a governance decision that should be made explicitly rather than inherited from an operational preference.

Reconciliation Architecture

Reconciliation is most effective when it is designed into the platform rather than added as an external process. Four architectural provisions make this possible. The first is the retention of independently derived control totals. Where a source system can produce a control total, the total should be captured at the point of extraction and stored with the extract. Capturing it later may be impossible, because the source data will have changed.

The second is the retention of the reconciliation key. Reconciliation requires the two figures to be matched on a common key. Where the platform transforms the key for example by mapping a source account number to an internal identifier the original key must be retained alongside the transformed one, or the reconciliation cannot be performed without re-deriving the mapping, which may not be possible for historical periods.

The third is the recording of reconciliation results in the metadata platform, so that result history is queryable and trends are visible. A reconciliation that produces the same variance every period should generate a remediation action, not a recurring explanation. The fourth is the ability to re-execute a reconciliation for a closed period. Circumstances arise in which a reconciliation must be repeated after the period has closed: a data correction is applied, a rule is revised, a supervisor asks a question. Where the reconciliation cannot be re-executed for a closed period, the institution cannot demonstrate the effect of the correction.

Worked Example: Investigating a Unexplained Variance

A monthly reconciliation between reported credit exposure and the general ledger's loan control account shows a variance of 0.4 per cent, within the structural expectation of 0.5 per cent but larger than the 0.1 per cent typical variance. The variance is within tolerance and the reconciliation passes. The investigation proceeds anyway, and the reasoning explains why reconciliation is treated as a diagnostic instrument rather than as a pass-fail gate.

The lineage graph is traversed from both figures to their common ancestor, which is the loan staging table. The paths diverge at the point where the reporting path applies a collateral netting rule that the ledger basis does not apply. The expected structural difference for that rule is quantified at 0.35 per cent, so the unexplained residual is 0.05 per cent against a typical 0.1 per cent that is, the variance is in fact smaller once the structural difference is properly attributed.

The exercise produces two improvements. The reconciliation definition is amended to encode the structural difference explicitly, so that future periods present only the genuine residual. And the rule inventory is examined to determine why the netting rule's contribution moved, which reveals a reference data update applied mid-period that changed the eligibility of a class of collateral. The second finding is a genuine control event: a reference data change altered a reported figure without the reporting function being notified, which is precisely the impact assessment failure described in Chapters 5 and 6.

Materiality, Tolerance and Threshold

Three related quantities are frequently conflated in reconciliation design and each has a different purpose. Materiality is a property of the reported figure: the magnitude at which a misstatement would be expected to influence a user of the report. It is set by the reporting framework and the institution's accounting policy, and it is not a design choice [58].

Tolerance is a property of the comparison: the difference within which two independently derived figures are regarded as representing the same underlying quantity. It is derived from the known structural differences between the bases being compared, and it should be documented as the sum of those differences rather than chosen from the observed history [54]. The threshold is a property of the control: the value at which the control reports a breach requiring an action. The threshold

should be set from tolerance for pass-fail purposes and from materiality for escalation purposes, so that a difference above tolerance but below materiality is investigated while a difference above materiality is escalated regardless. Setting the threshold from the recent observed difference, which is a common practice, produces a control that ratifies whatever the current process happens to produce.

Reconciliation Frequency and the Cost of Latency

The frequency of a reconciliation determines how long a defect can propagate before it is detected, and the cost of that latency is the cost of the corrections it necessitates. A daily reconciliation that compares the day's movements detects a systematic error after one day's data has been affected. A monthly reconciliation detects it after a month, by which time the correction must be applied to an aggregated figure whose components have been transformed and possibly reported. A quarterly reconciliation detects it after a quarter, and the correction may now affect a submitted return and require a restatement.

The design principle is therefore that reconciliations should be performed at the frequency of the fastest process that can introduce the defect they detect. A controls total published daily should be reconciled daily even where the obligation it protects is quarterly, because the earlier the difference is visible the cheaper its resolution.

Reconciliation Evidence

A reconciliation produces three artefacts, and all three are required for the control to be effective. The first is the result: the two figures, the difference, the tolerance, the threshold and the pass or fail determination. The second is the basis: the specification of what is being compared, including the structural differences that are expected and the treatment of each. The third is the disposition: for a failing reconciliation, the investigation performed, the cause identified and the action taken, signed by the person accountable.

The first artefact without the second cannot be interpreted by a reviewer, because a difference of a given size passes or fails depending on the basis. The first two without the third describe a problem without recording that it was addressed. Examinations ask for all three, and an institution that retains only the first has evidence that a comparison was performed and none that it was understood [54], [52].

Reconciling Transformation and Reporting Layers

A specific reconciliation class compares the aggregation layer against the reporting layer, and its purpose is to detect processing defects introduced between them rather than data defects inherited from the sources.

Its design takes the base aggregation's control total and compares it against the sum of the corresponding reporting-layer figures, expressed in the reporting framework's sign and unit conventions. The characteristic differences it must accommodate are sign conventions, unit scaling, treatment of eliminations and the inclusion or exclusion of adjustment journals. Where these are encoded as expected adjustments, the residual is a genuine control signal about the pipeline.

The reconciliation is valuable because the reporting layer usually contains the most transformation per unit of data and the fewest controls per unit of transformation, since the controls are concentrated at the earlier stages where the data is understood to be less trustworthy. Figure Discrepancy Investigation and Resolution traces the controlled path from an identified variance through investigation and disposition to a signed resolution, and shows the point at which each step is evidenced.

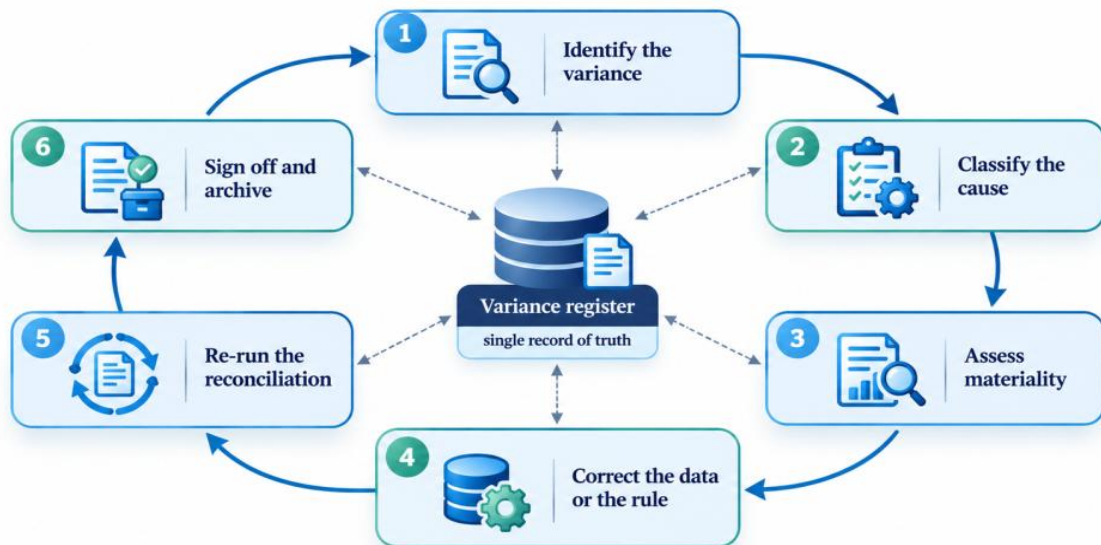


Figure 9.3 Discrepancy Investigation and Resolution

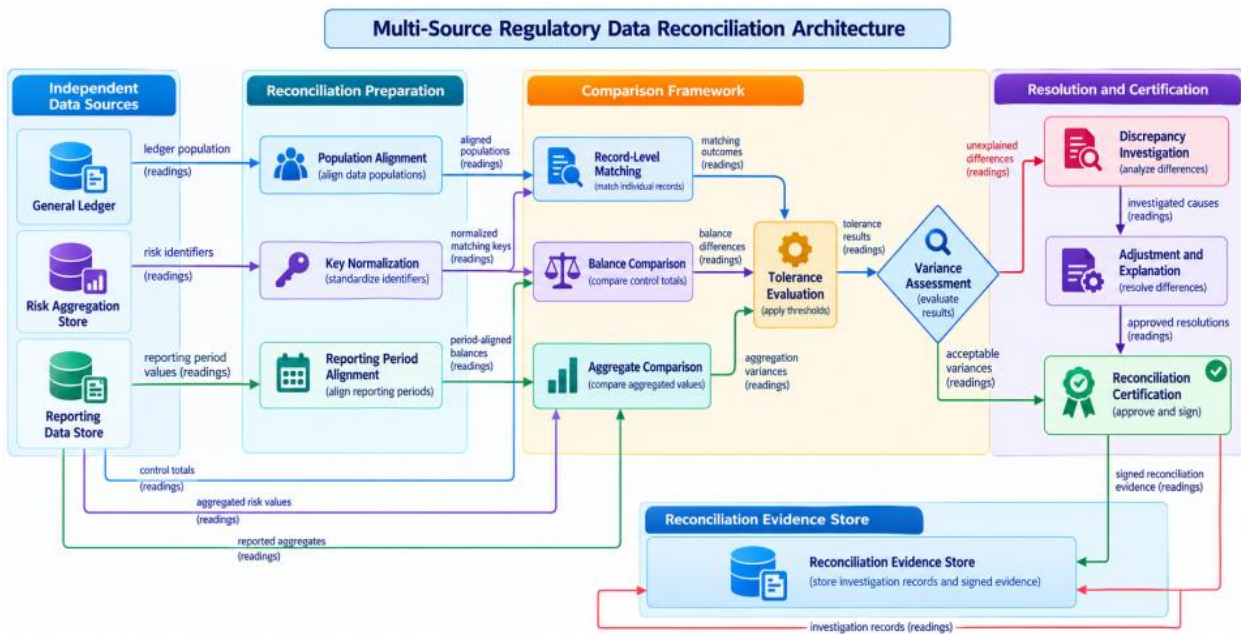


Figure 9.4 Multi-Source Regulatory Data Reconciliation, Variance Assessment, and Certification Architecture

An enterprise architecture for reconciling regulatory data obtained from independent sources, including the general ledger, risk aggregation store, and reporting data store. The Reconciliation Preparation layer aligns source populations, normalizes key identifiers, and establishes reporting periods to ensure that datasets are comparable. These preparation activities provide aligned populations, normalized matching keys, and period-aligned balances for the subsequent comparison process. The Comparison Framework performs record-level matching, balance comparison, and aggregate comparison to identify differences between source datasets. The tolerance evaluation component determines whether observed differences fall within defined acceptable thresholds, while variance assessment evaluates the significance of the remaining discrepancies. Variances that exceed acceptable limits are directed to the Resolution and Certification layer, where discrepancy investigation identifies potential causes, adjustment and explanation activities document resolutions, and reconciliation certification records the approved status of the results. The Reconciliation Evidence Store preserves investigation records and signed evidence associated with reconciliation activities. This evidence supports traceability from source data through comparison, variance evaluation, and final certification. By combining multi-source alignment, layered comparison, tolerance-based assessment, discrepancy resolution, and evidence retention, the architecture supports controlled reconciliation and strengthens the reliability and auditability of regulatory reporting data.

CHAPTER 10

Exception Management, Data Remediation, and Audit Evidence

A control framework that detects defects and does nothing with them provides no assurance. This chapter addresses the operational machinery through which detected defects are classified, routed, remediated, verified and closed, and through which the record of that activity is retained as evidence. It treats exception management as a production process with service levels, queues and ageing, rather than as a residual activity performed by whoever is available. The chapter's central claim is that the treatment of exceptions reveals the maturity of a data organisation more accurately than any other observable. A mature organisation has a single exception queue, classifications that determine ownership and urgency, defined remediation paths and a closed-loop verification step. An immature organisation has exceptions distributed across email, spreadsheets and individual memory, with no reliable count of what is outstanding.

The Exception Lifecycle

Figure Exception Lifecycle presents the exception lifecycle as six stages. The lifecycle is circular in the sense that closure generates information that feeds back into rule design, but each individual exception traverses it once.

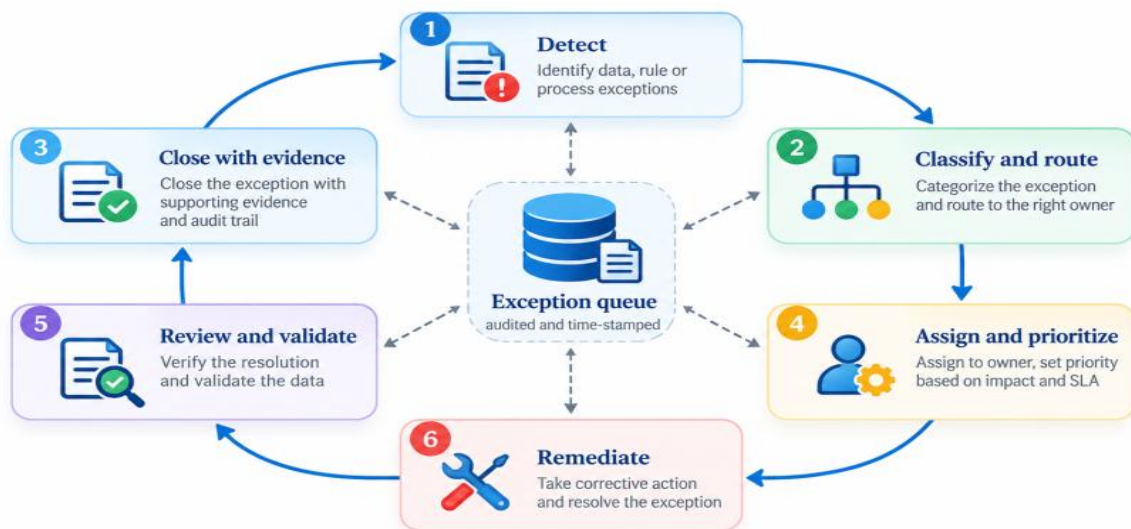


Figure 10.1 Exception Lifecycle

Detection

Exceptions arise from rule failures, reconciliation variances, manual observations and external notifications. The first two are systematic and produce exceptions automatically. The last two are unsystematic and are the reason a single queue is necessary: an exception reported by a person must enter the same process as one detected by a rule, or it will be tracked separately and inconsistently. A design point worth stating: an exception should be created as a record with a stable identity at the moment of detection. Where exceptions are represented as rows in a report rather than as records in a queue, they cannot be assigned, aged or tracked, and the organisation cannot answer the question of how many exceptions are open.

Classification and Routing

Classification determines what the exception is, which determines who owns it and how urgently it must be treated. Figure Exception Classification Scheme presents a four-family classification scheme.

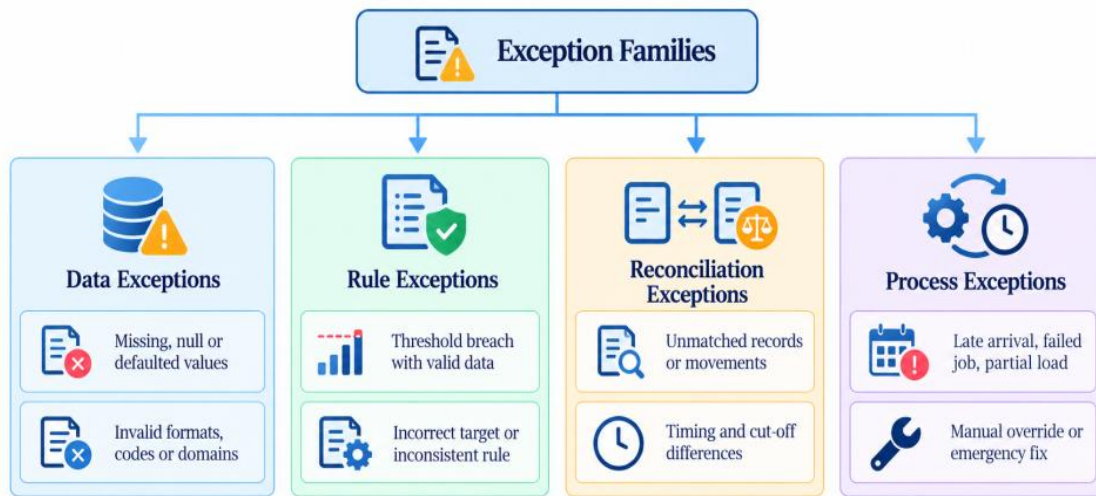


Figure 10.2 Exception Classification Scheme

Data exceptions arise from defects in values: missing, null, defaulted, malformed or out-of-domain values, and values that fail referential resolution. Ownership normally lies with the data steward for the affected domain, and remediation is at the data or at the source that produced it. Rule exceptions arise where valid data breaches a rule threshold, or where a rule itself is found to be inconsistent with policy. The first case requires an investigation to determine whether the data or the threshold is wrong. The second requires the rule lifecycle described in Chapter 8. Reconciliation exceptions arise where two independently produced figures disagree beyond tolerance. Ownership is shared between the reporting function and the owners of the two figures, and resolution requires the root-cause analysis described in Chapter 9.

Process exceptions arise from the operation of the pipeline rather than from its content: a late feed, a failed job, a partial load, a manual override. These are the exceptions most likely to be treated informally, and the most important to treat formally, because they are the mechanism by which defects enter an otherwise sound process. Classification should be assigned automatically where the detection mechanism implies it, and manually with an explicit value where it does not. The classification should be immutable once set, with reclassification recorded as a change with a justification, so that the history of how an exception was understood is preserved.

Assignment and Prioritisation

Ownership assignment determines accountability. Every exception must have a named owner rather than a team, because team ownership is a mechanism by which responsibility dissipates. Prioritisation should follow from the classification and from materiality rather than from arrival order or from the seniority of the person who raised it. A practical prioritisation scheme combines the consequence of the exception if unresolved with the time available before the exception affects a reporting deadline. The second element is what gives exceptions their urgency, and its inclusion is what makes the queue tractable under deadline pressure.

Remediation

Remediation corrects the condition that produced the exception. Four remediation paths are available, and the choice between them is a governance decision that should be documented.

- Correct the data is appropriate where the data is wrong and the rule is right. It is the most common path and the most laborious where corrections must be applied in a source system rather than in the platform.
- Correct the rule is appropriate where the rule is wrong. It requires the rule change lifecycle described in Chapter 8, including impact assessment and testing.
- Remove the record is appropriate where the record should not be in the population at all. It is the path most prone to misuse, because it can be used to make an exception disappear rather than to resolve it. Its use should require approval and should be recorded with a justification and the identity of the approver.

- Accept with a documented waiver is appropriate where the exception cannot be remediated before the deadline and its consequence has been assessed and accepted. Waivers must be time-limited, approved at the appropriate level and recorded, so that they do not become permanent by inaction.

Verification and Closure

Verification establishes that the remediation worked. It should be performed by a step that is independent of the remediation, even if that independence is achieved only by a separate automated re-execution of the detecting control. Where the detecting control is re-executed, the exception should close automatically on a passing result. Where it is not, the remediation should be verified by inspection of the corrected data. In either case, closure should record who verified it, when, and by what evidence. Closure should also record the root cause classification. This is the element that converts the exception process from a closed loop per exception into a learning system across exceptions. Where root causes are recorded consistently, the institution can analyse them and identify the recurring mechanisms that generate defects. That analysis is the most reliable guide to where structural investment should be made.

Remediation Workflow and Evidence

Figure Remediation Workflow and Audit Evidence Trail presents the remediation workflow as a swimlane diagram with the accountability at each step made explicit. The diagram is presented in this form deliberately, because the most common failure in exception management is not a missing step but an unassigned one.

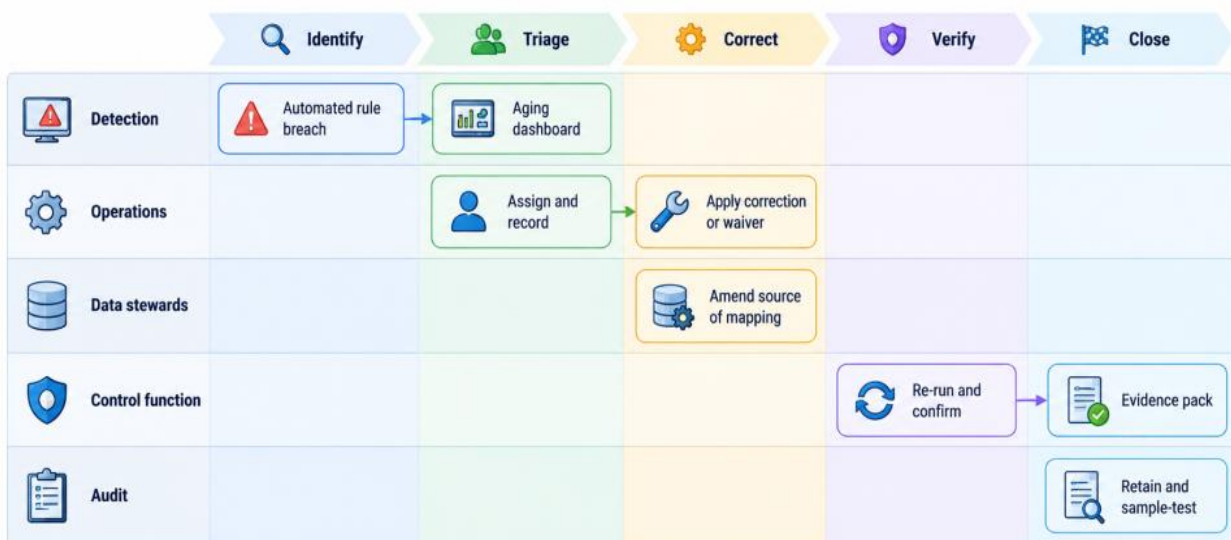


Figure 10.3 Remediation Workflow and Audit Evidence Trail

The Evidence Requirement

Every step in the workflow produces evidence, and the evidence must be retained in a form that can be produced during an examination. Six artefacts are characteristic. The exception record establishes that the exception was detected, when, by which control and with what result. The assignment record establishes who owned it. The investigation record establishes what analysis was performed and what conclusion was reached. The remediation record establishes what was changed. The verification record establishes that the change was effective. The closure record establishes who accepted the closure and on what basis. In practice these are frequently implemented as fields on a single record with an audit history rather than as separate artefacts. The implementation choice is immaterial; what matters is that each of the six facts is captured and retained.

Retention Periods

Exception records are evidence and must be retained for the period required by the institution's records policy and by the applicable supervisory regime. The retention period for exception records is frequently shorter than for reported data, on the reasoning that exceptions are operational rather than reportable. The reasoning is unsound. The exception record is the evidence that a known defect was identified and handled, and its absence during an examination of a past period leaves the institution unable to demonstrate that it acted.

Exception Metrics and Their Interpretation

Metrics make the exception process manageable, and their interpretation matters as much as their measurement.

Open exception count is the basic workload measure, useful primarily in trend. A rising count indicates that detection is outrunning remediation, which is a resource or process problem rather than a quality problem. Ageing distribution is more informative than the count, because it distinguishes a stable queue that is being worked from a queue that has stalled. A queue with a growing tail of old exceptions is a queue in which difficult cases are being avoided, and the tail is where the material problems reside.

Time to remediate, segmented by classification, indicates whether the ownership model is working. Long times to remediate a data exception indicate that the data steward is not empowered to act. Long times to remediate a reconciliation exception indicate that the investigation capability described in Chapter 6 is inadequate. Recurrence rate is the proportion of exceptions whose root cause has been recorded before. A high recurrence rate is the clearest available evidence that remediation has been applied to symptoms. It is the metric that most directly measures whether the organisation is learning.

Waiver rate is the proportion of exceptions closed by waiver rather than by remediation. A rising waiver rate under deadline pressure is normal and expected. A persistently high waiver rate indicates that the control thresholds or the remediation capacity are misaligned with the obligation. False positive rate is the proportion of exceptions that close without any underlying defect. It measures the precision of the detecting controls. A high false positive rate is corrosive, because it consumes remediation capacity and trains reviewers to discount detections.

The Trade-Off Between Detection and Capacity

There is an unavoidable relationship between the number of controls, the sensitivity of their thresholds and the remediation capacity available to handle their output. An institution that increases detection without increasing capacity produces a queue that grows until it is abandoned, and an abandoned queue is worse than no queue because it creates the appearance of control without the substance. The practical approach is to sequence detection expansion against capacity. Controls are introduced at a sensitivity the organisation can service, and sensitivity is tightened as the underlying quality improves and capacity is freed. This is the same discipline described for threshold management in Chapter 4, applied to the control framework as a whole rather than to individual rules.

Manual Interventions and Their Governance

Manual intervention is a necessary feature of any reporting process. Adjustments must sometimes be applied that no rule anticipates, and the alternative refusing to correct a known error is worse. The governance question is not whether manual interventions occur but whether they are controlled. Four controls should apply to every manual intervention. The intervention should be recorded as an artefact with the period, the reporting line affected, the amount, the reason and the person applying it. It should be approved by an authority appropriate to its size. It should carry forward into the following period, so that a recurring adjustment becomes visible as a pattern and is investigated rather than accepted indefinitely. And it should be reconciled in the following period, so that an adjustment applied in one period is reversed or confirmed rather than silently accumulating. The most common governance failure is the accumulation of adjustments that are applied once and never revisited. An adjustment that is genuinely a correction of a one-off event is reversed or superseded naturally. An adjustment that corrects a systematic defect will recur, and its recurrence is the signal that the underlying defect requires remediation. Where adjustments are recorded but not analysed, that signal is not read.

Segregation of Duties in Exception Handling

The control environment requires separation between the person who detects, the person who remediates and the person who verifies. In small teams, complete separation is impractical, and the compensating control is documented review by a person outside the immediate process. The separation that matters most is between the person who can change a rule or a threshold and the person who reviews the resulting exception volume. Where the same person can adjust a threshold to reduce the exceptions it generates, the control is ineffective regardless of how well it is documented.

Interaction with the Reporting Calendar

Exception management must be integrated with the reporting calendar, because the two have conflicting time horizons. The exception process operates continuously; the reporting process operates to immovable deadlines. The integration point is the cut-off. Before each reporting deadline, the open exception queue for the affected population must be reviewed, and each open exception must be dispositioned: remediated, waived with approval, or carried forward with a documented effect on the reported figure. This review is a control in its own right and should be evidenced. Its failure mode is well known: under deadline pressure the review is abbreviated, exceptions are waived in bulk without individual assessment, and the reported figure is released with an unquantified population of known defects. The evidence of a properly conducted review a dated disposition for each open exception is the artefact that distinguishes the two cases.

Worked Example: Dispositioning the Queue Before a Deadline

Six working days before a submission deadline, the exception queue for the affected population holds sixty-two open items. The disposition exercise classifies and assigns each one. Forty-one are data exceptions of a familiar type: missing optional attributes on a small number of low-value records. These are assessed, the aggregate effect on the reported figure is computed and found immaterial, and they are waived with the individual values recorded and the waiver approved by the reporting owner. Twelve are reconciliation exceptions whose causes have been identified and whose corrections are in progress; these are tracked to completion, and the reporting owner is informed that the return must not be released before they close.

Five are data exceptions whose corrections require a source system change outside the reporting function's control; these are carried forward with a quantified effect on the reported figure, a documented explanation and a scheduled remediation. Two are rule exceptions that reveal a rule inconsistent with the current framework version; these cannot be remediated before the deadline and are waived with an explanation, with the rule change scheduled. One is a process exception whose cause is unknown; it is escalated because an unexplained process exception is a risk that cannot be quantified, and the escalation produces a root cause a partially failed extraction before the deadline. The artefact produced by the exercise is a dated disposition for each of the sixty-two items, which is the evidence that the pre-deadline review was conducted rather than abbreviated. The fifty-four waived and carried items are on the following period's review list, so that the accumulation of waivers becomes visible as a pattern rather than remaining invisible.

Root-Cause Analysis That Changes Something

Root-cause analysis is frequently performed as an explanation exercise and rarely as a corrective one. The distinction is visible in the artefact: an analysis that concludes with a description of how the defect arose has explained it, while an analysis that concludes with a change to a rule, a control, a source specification or a process has prevented it [6], [10].

A structure that produces the second outcome proceeds through four questions. What was observed, stated in terms of the control result rather than the symptom? Through which mechanism did the defect produce that observation, traced through the pipeline rather than inferred? Why did the mechanism operate, at the level of the design decision or the process step rather than the individual action? And what change removes the mechanism, expressed as a specific alteration with an owner and a date?

The third question is the one that distinguishes the analysis from an investigation. Its answers fall into a small number of recurring classes: an expectation that was never defined, a control that was defined but not executed, a control that was executed but whose threshold was mis-set, a reference data gap, a source change applied without notice, or a manual step whose variance was not anticipated. Naming the class allows the institution to see whether it is encountering the same class repeatedly, which is the evidence that a systemic remediation is required rather than a further individual correction.

Designing the Exception Record

The exception record is the unit of work in exception management, and its field set determines which questions the institution can answer about its own remediation history. A conforming record carries: a unique identifier; the classification of the exception; the control rule that produced it, with its version; the affected population and the reported data points at risk; the severity, derived from the materiality of the figures at risk rather than assigned by the raiser; the detection timestamp and the period to which it relates; the state, drawn from a defined transition set with no undefined

states; the assigned owner; the analysis and the cause where established; the resolution and its basis; the waiver or correction decision with its approving actor; and the closure timestamp. The field set is deliberately close to the record structure that an examination will request, because producing it as a by-product of routine operation is inexpensive and reconstructing it during an examination is not.

Measuring the Health of the Remediation Process

Four indicators, computed from the exception records rather than maintained separately, describe whether the remediation process is working. The age distribution of open exceptions shows whether work is progressing or accumulating. The proportion of exceptions arising from previously identified causes shows whether root-cause analysis is producing prevention.

The proportion closed by waiver shows whether the institution is correcting its data or tolerating its defects, and a rising waiver rate is the earliest visible signal of erosion in the control environment. The median time from detection to resolution, by severity, shows whether the process responds in proportion to the materiality of what is at risk. None of the four requires additional instrumentation. All four are queries against the exception store, which is the practical argument for keeping exceptions as governed records rather than as rows in a spreadsheet circulated by mail.

The Boundary Between Data Remediation and Source Remediation

A recurring decision is whether a defect is corrected in the reporting platform or returned to the source for correction. Correcting in the platform is faster and within the reporting function's control, and it makes the platform's data differ from the source of record, which is itself a control weakness because subsequent comparisons will show a difference. Returning to the source is slower, requires the source owner's cooperation and results in a single correct value that all consumers can use.

The decision rule that follows is to correct in the platform only where the correction is required for a specific reporting obligation and is recorded with its basis, and to return to the source wherever the defect affects more than one consumer or where the correct value is knowable only at origin. Corrections made in the platform without a recorded basis are indistinguishable from errors introduced by the platform, which is why the platform correction must always carry its evidence [37].

Evidence Retention: What, Where and For How Long

The retention question is answered by the obligation rather than by the platform's storage preference, and the answer has three parts. What must be retained is determined by what must be reproducible. For each reported figure the institution must retain the source data as landed, the code and rule versions in force, the reference data versions in force, the configuration in force, the intermediate results sufficient to reproduce the figure, the control results and their thresholds, the exceptions raised and their dispositions, the adjustments applied and their basis, and the release record with its approval. Retaining the reported figure alone satisfies none of these.

Where it must be retained is determined by accessibility rather than by cost. An immutable archive on slow storage satisfies retention and fails accessibility, because the first examination question will require a retrieval that takes days. The practical arrangement keeps the recent periods on accessible storage and archives older periods in a form that can be restored within a stated time, with the restoration tested rather than assumed. How long it must be retained is determined by the longest applicable requirement, which is usually the limitation period for supervisory action rather than the reporting framework's own retention rule. The policy should state the period and the basis for it, because the retention of data beyond the requirement is itself a privacy exposure and the retention of data below it is a compliance failure [45], [55].

CHAPTER 11

Regulatory Reporting Data Models and Information Delivery

The preceding chapters addressed the movement, quality, meaning and control of data. This chapter addresses its final representation: the data models through which regulated information is delivered to the reporting templates, and the design principles that determine whether those models support the reporting obligation durably. The central argument is that the reporting data model should be designed against the semantic requirements of the obligation rather than against the physical layout of the template. A model designed around the template reproduces the template's structure, and when the template changes as regulatory templates change frequently the model must be rebuilt. A model designed around the underlying economic and legal semantics can be projected into successive versions of a template with changes confined to the projection layer.

The Layered Reporting Data Model

Figure Regulatory Reporting Data Model Layers presents the reporting data model as five layers. The layering is not incidental; it is the mechanism by which the model absorbs change.

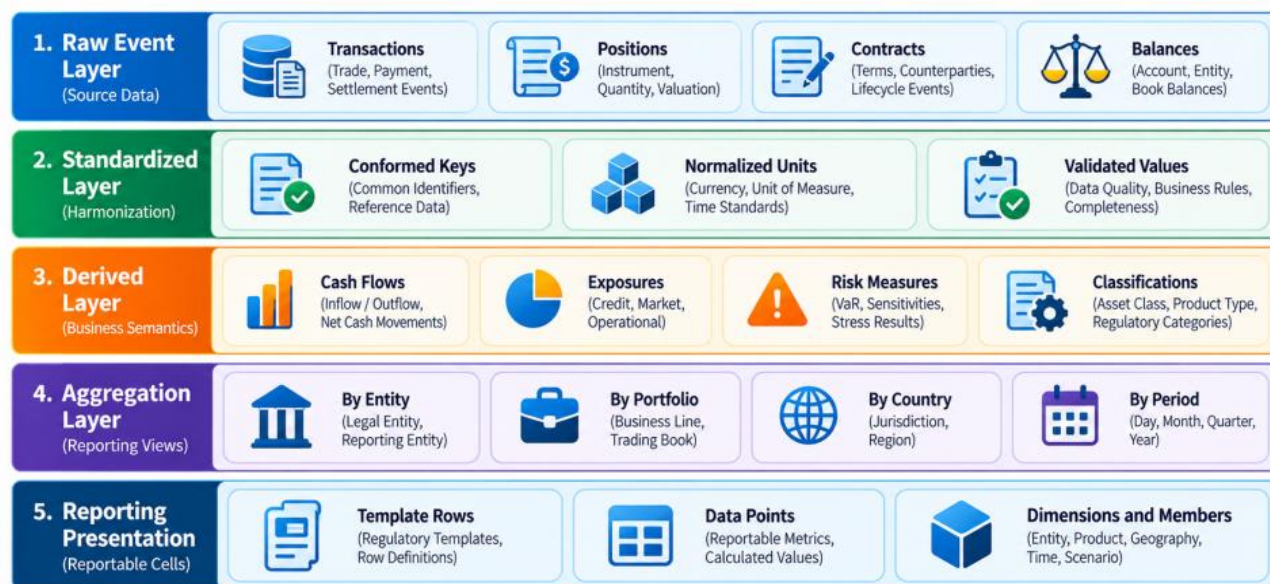


Figure 11.1 Regulatory Reporting Data Model Layers

The raw event layer holds data as received: transactions, positions, contracts and balances, in the form in which the source produced them. It is the landing zone described in Chapter 7 and is normally immutable. The standardized layer holds the same data with conformed keys, normalized units and validated values. It is the output of the ingestion and transformation stages described in Chapters 7 and 8. Its defining property is that its semantics are consistent across sources: a monetary amount in this layer is in a known unit, an entity reference in this layer refers to a known entity identity, a date in this layer has known period semantics.

The derived layer holds values that are computed rather than observed: cash flows projected from contractual terms, exposures after netting and collateral, risk measures, and regulatory classifications. Its defining property is that every value in it is traceable to a rule and, through that rule, to the standardized layer. The aggregation layer holds summaries: by entity, by portfolio, by country, by period and by every other dimension the reporting obligations require. Its defining

property is that it is reproducible from the derived layer by a defined aggregation specification. The reporting presentation layer holds the data in the structure the template requires: rows, columns, data points and dimensional members. Its defining property is that it contains no logic beyond the mapping from the aggregation layer to the template structure.

Why the Boundary Between Derivation and Presentation Matters

The boundary between the derived and presentation layers is the most consequential design decision in a reporting data model, and it is frequently placed in the wrong position. Where the boundary is placed too low that is, where derivation is embedded in the presentation layer the reporting template contains logic. A template cell that computes a ratio from two other cells is embedding derivation in presentation. The consequence is that the same ratio computed for a different report is computed by a different expression, and the two may disagree. Regulatory reporting is replete with examples of figures that should be identical across returns but are not, because the logic producing them was implemented independently in each return. Where the boundary is placed correctly, every derived value is computed once in the derived layer and every presentation cell is a reference to it. Where a value is genuinely specific to one template a presentation ratio that exists only because the template requires it the derivation still lives in the derived layer, so that it is inventoried, versioned and tested by the same process as every other rule. The practical test is whether the model can answer the question where is the rule that computes this reported cell? with a single reference to the rule inventory. Where the answer requires reading template-specific code, the boundary is misplaced.

Granularity and the Finest-Common-Denominator Principle

The aggregation layer must be built at a granularity at least as fine as the finest obligation it serves, because aggregation is reversible while disaggregation is not. The principle is straightforward and its violation is common. A model built at entity level cannot produce an instrument-level disclosure. A model built at monthly frequency cannot produce a daily series. Recovering the finer granularity after the fact requires returning to the source, and the source may not retain the detail for the historical period. The countervailing consideration is cost. Building at the finest granularity increases storage, processing time and control volume. The practical resolution is to build at the finest granularity that any current or reasonably anticipated obligation requires, and to document the granularity decision with its rationale so that the constraint is visible when a new obligation is assessed.

Dimensional Conforming

Figure Target-State Mapping to Reporting Data Points presents the dimensional structure of the reporting mart. The principle applied is conformed dimensions: dimensions that are shared across reporting subjects are defined once and referenced consistently.

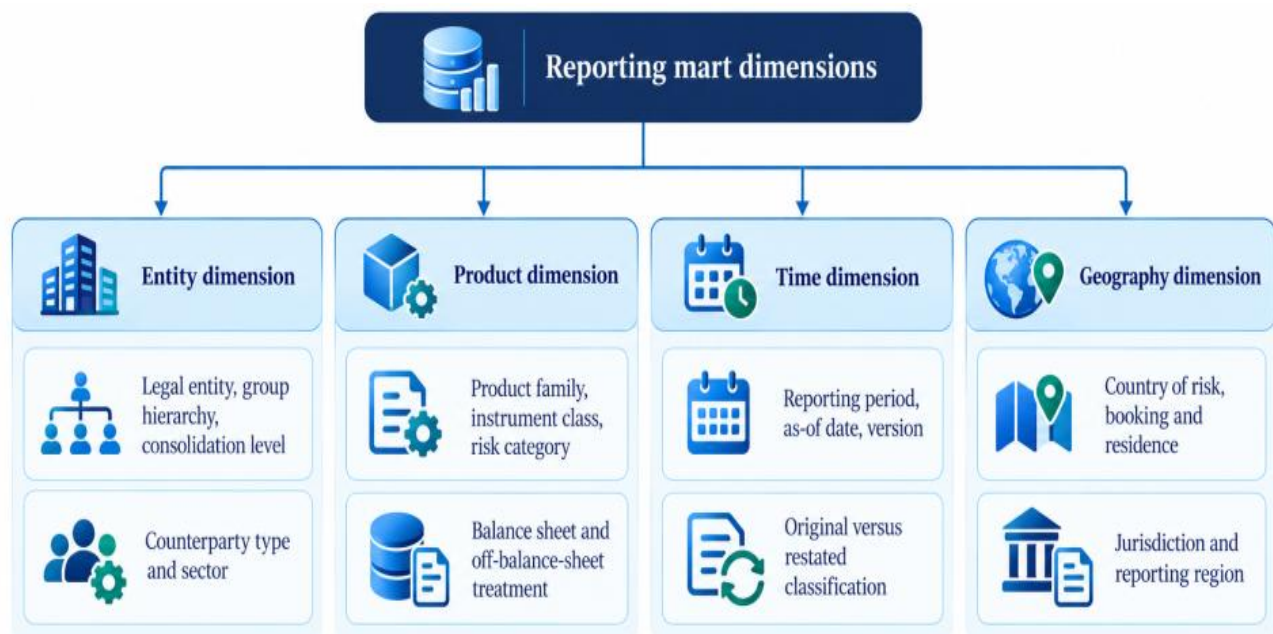


Figure 11.2 Dimensional Structure of the Reporting Mart

- **Entity dimension.** The entity dimension describes the legal entities and organisational units about which the institution reports. It must support the group hierarchy, the consolidation level, the regulatory scope of consolidation and the mapping between the internal entity identifier and external identifier schemes. Its most demanding requirement is temporal: the hierarchy as at a date, which requires the relationship structure described in Chapter 5.
- **Product dimension.** The product dimension describes what is being reported: the instrument class, the product family, the risk category and the balance-sheet treatment. Its most demanding requirement is the mapping between the institution's operational product taxonomy and the regulatory taxonomy, which is a conforming rule set governed as described in Chapter 8. The residual cases where no clean mapping exists are the ones that generate reconciliation friction and supervisory questions.
- **Time dimension.** The time dimension describes the period, the as-of date and the version. The version element is essential and frequently absent. A return for a period may be produced, revised and reproduced, and the ability to distinguish versions is what allows the institution to state which figures were reported and which were subsequently restated.
- **Geography dimension.** The geography dimension describes where the exposure, the entity or the booking resides. The multiplicity of geographic concepts is a chronic source of reporting defect: country of risk, country of booking, country of residence and country of incorporation are distinct concepts, and a model that holds a single country attribute will be forced to choose one, which means the others are unavailable for obligations that require them.

Slowly Changing and Rapidly Changing Dimensions

Dimensional modelling has long distinguished slowly changing dimensions from rapidly changing ones, and the distinction has practical consequences for regulatory reporting. Attributes that change slowly an entity's name, an instrument's classification should be modelled with full history, so that a report produced for a past period reflects the attribute values in force at that time. This is the standard slowly changing dimension treatment, and it is a requirement rather than a refinement in a regulatory context. Attributes that change rapidly should be modelled with the change captured, so that the change can be explained. A credit rating that changes frequently is not merely a slowly changing attribute with many versions; the dates of change matter, because the reporting treatment may depend on the rating in force at the reporting date rather than the rating currently held. The practical implementation adds effective dating to the attribute rather than merely versioning the record.

Mapping Targets to Reporting Data Points

Figure Dimensional Structure of the Reporting Mart presents the mapping of source attributes to a specific reporting template cell. The mapping is the artefact that answers the traceability question at the reporting layer, and it should be governed with the same discipline applied to transformation rules.

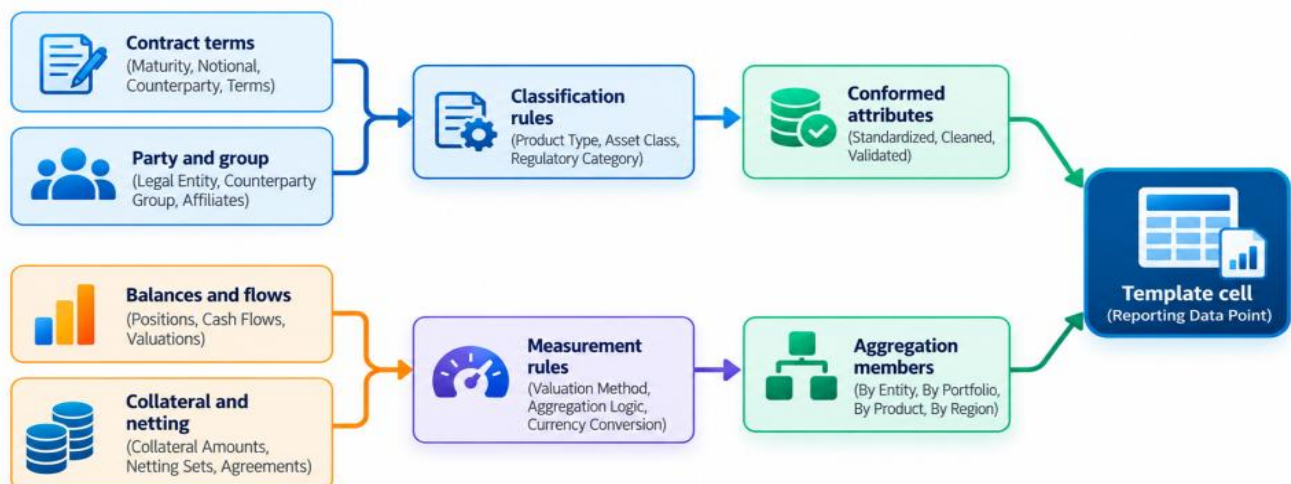


Figure 11.3 Target-State Mapping to Reporting Data Points

The Mapping Specification

A complete mapping specification identifies the target data point, the source attribute or attributes, the transformation applied, the dimensional members used for filtering and grouping, the sign convention, the unit and the control that verifies the result. Four of these components deserve emphasis because they are frequent sources of restatement. The sign convention is a persistent source of error because source systems disagree. A liability recorded as a positive number in one system and as a negative number in another will produce a reported total that is either correct or double-counted, depending on which convention the aggregation assumes. The convention should be declared in the mapping and tested by an arithmetic control rather than assumed.

The unit and scale must be declared explicitly. Amounts recorded in thousands in one source and units in another produce discrepancies of three orders of magnitude that are visible immediately when they occur and catastrophic when they are masked by a compensating error elsewhere. Unit declaration in the mapping, with automatic conversion to a canonical unit in the standardized layer, eliminates the class of error. The dimensional members determine which records are included. A mapping that specifies the target as an attribute and the filter as an unstated convention will be implemented differently by different engineers. Where the filter is explicit in the mapping specification, the population is reviewable.

The sign and rounding rules applied at the reporting layer deserve attention because the reporting obligation frequently specifies them. A total computed by summing rounded components may differ from a total computed by summing exact components and then rounding. The difference is small and it is visible to a supervisor who reconciles the components to the total in a published return. The mapping specification should state which treatment applies, and the control should verify that the published components sum to the published total.

Template Versioning and Taxonomy Release

Reporting templates change, and their change is managed through taxonomies and release cycles. The architectural requirement is that the mapping from the aggregation layer to the template be versioned in step with the template. The practical arrangement maintains a mapping version per taxonomy release, with the release identity recorded with every generated return. When a new release is adopted, the mapping is revised, tested against the previous period's data, and the differences are quantified and explained. The last step is the one that converts a taxonomy migration from a surprise into a managed change: applying the new mapping to the prior period's data and reconciling the result against the return as originally reported.

Presentation Logic and Format Compliance

The presentation layer's obligations extend beyond values. Reporting frameworks specify the format, the dimensional structure, the units, the precision and the rounding of the information transmitted. Where the framework is codified in a machine-readable taxonomy, the format requirements are enforceable automatically. Where it is specified only in prose, they must be encoded as validation rules. The most reliable architecture treats the presentation as a rendering step rather than as a transformation. The values are computed in the derived and aggregation layers; the presentation layer arranges them into the required structure and serialises them. Where the presentation layer performs calculation, the calculation is invisible to the rule inventory and therefore to the impact assessment process.

Information Delivery Patterns

The final question of this chapter is how the reporting data model delivers information to its consumers. Four patterns are used, and their selection determines much of the operational character of the reporting platform. Scheduled batch delivery produces the full set of reportable data at the conclusion of the reporting cycle and delivers it as a bounded artefact. It is the dominant pattern for periodic returns and its principal virtue is that it produces a discrete, versioned, reviewable artefact. Regulatory gateway submission delivers the reportable data to an external authority through a defined interface, with acknowledgements. It introduces the receipt and error-handling semantics discussed in Chapter 12.

Analytical delivery exposes the reporting data model to internal analytical consumers finance, risk, treasury who require it for their own analysis. This pattern is valuable because it converts the reporting model into a shared enterprise asset and creates an additional class of consumer whose use validates the model's definitions. It also introduces a conflict: analytical consumers require flexibility and latency that the reporting obligation does not, and accommodating them can compromise

the controls required for reporting. The practical resolution is to serve analytical consumers from a distinct view that shares the model but not the control gates. Supervisory ad hoc delivery provides data in response to a specific supervisory request, frequently at short notice and in a non-standard structure. The capability requirement is that the underlying data model be granular enough and traceable enough to answer questions that were not anticipated. This is the pattern that most directly rewards the investments described in Chapters 2 and 6.

Worked Example: Where the Boundary Should Sit

A template requires a reported ratio computed from two other reported cells. Two placements are possible. In the first, the ratio is computed in the presentation layer as a division of the two rendered cells. It is fast to deliver. It is also invisible to the rule inventory, absent from the lineage graph and impossible to test independently. When a second template requires the same ratio, it is computed again by a different expression, and the two are found to differ because one uses unrounded inputs and the other uses the rounded values displayed in the cells. In the second, the ratio is derived in the derived layer as a governed rule with its own identity, version, owner and test cases. The presentation layer references it. When the second template requires the same ratio, it references the same rule, and the two figures agree by construction. The tested behaviour of the rule includes the rounding question, which is resolved once, explicitly, in accordance with the framework's specification. The second placement costs perhaps a day of additional effort on first delivery and removes a class of defect that is both embarrassing and recurring.

Reconciliation Between Reporting Obligations

A distinct control class arises where the same underlying quantity appears in more than one obligation. The obligation-specific figures should be reconcilable to each other, and the reconciliation is a control on the mapping specifications rather than on the data. Its operation is straightforward: for each quantity that appears in more than one obligation, compare the figures as reported, express the difference in terms of the definitional differences between the obligations, and require the difference to be explained entirely by those definitions. An unexplained difference indicates that one of the mappings is incorrect, or that the two obligations are consuming different data versions, or that a filter condition differs between them. The control is inexpensive where mapping specifications are governed as described earlier and impossible where they are not, because establishing which obligations share a quantity requires a query over the mapping register rather than a review.

The Reporting Data Model: Conformed, Not Bespoke

A distinct reporting data model for each return produces high first-delivery speed and compounding maintenance cost, because a change in a shared concept must be applied in as many places as there are returns. The alternative is a conformed reporting model: a small set of governed entities and measures that any reporting projection draws upon, with the framework-specific aspects taxonomy element, sign convention, unit, template placement held as governed metadata attached to the projection rather than as structure in the model [21]. The conformed model is more expensive to establish and it is the reason a second return costs a fraction of the first. The practical governance requirement is that the conformed model have an admission process, as described earlier, so that it does not become an accumulation of convenience. Its measures should be defined once, with a stated derivation and an owner, and its framework-specific attributes should be maintained as mappings rather than as columns.

Representing Adjustment Journals

Adjustment journals are the mechanism by which a knowledgeable individual alters a reported figure, and their treatment determines whether the institution can explain its reported numbers. Three design requirements follow. Every adjustment must be represented as data rather than applied as an edit to the reported figure, so that the reported figure remains a function of its inputs. Every adjustment must carry its basis: the identity of the person authorising it, the reason, the period to which it applies and the evidence supporting it. And every adjustment must be visible in the reconciliation between the pre-adjustment and post-adjustment figures, so that the effect of the adjustment layer can be quantified for any reported cell. Where adjustments are applied as direct edits, the platform's lineage describes a derivation that did not produce the reported figure, and the institution's explanation of its report is incomplete in exactly the area where a reviewer will concentrate [54], [49].

Disclosure and Reconciliation to Financial Statements

The figures reported to a supervisor should reconcile to the figures disclosed in the institution's financial statements, and the reconciliation is a control on both. Its design compares each reported figure with its counterpart in the disclosed financial data, expressing the difference in terms of the definitional differences between the regulatory and accounting bases: scope differences, valuation differences, consolidation differences and timing differences. The result is a bridge, and the bridge's value is that it can be reviewed for consistency by people responsible for the financial statements as well as by the reporting function. An unexplained residual in the bridge indicates either that a mapping is incorrect or that a figure has been adjusted without the adjustment being reflected in both places. Both are material findings, and the reconciliation is the least expensive way to detect them [58].

Documentation of a Reporting Data Point

The documentation of a reported data point is the artefact that a supervisory reviewer is most likely to request, and its content follows from the questions such a review asks. The documentation should state the data point's identity within its framework and version; its definition in the framework's own terms; its internal derivation, expressed as the rule or rules that produce it, with versions; its source attributes with their systems of record; its controlling dimensions and the controls that assert them; its owner; and its known limitations, including any population that the institution cannot fully source and the compensating control applied. Artefacts that state only the first three are descriptions. The last provision the known limitations and the compensating control is the one that distinguishes a candid document from a favourable one, and it is the provision that makes the document useful during a review, because a limitation disclosed and controlled is a managed risk while the same limitation discovered by the reviewer is a finding [49].

Serving Ad Hoc Supervisory Requests

Ad hoc supervisory requests are the harshest test of a reporting architecture because they arrive without notice, without a specification and frequently at a granularity the institution does not maintain. Four architectural properties determine the response time. Where a governed aggregation layer exists at the granularity required, the request is answered by projection. Where it does not but the source data is retained at that granularity, the request is answered by a build against retained data. Where the retained data is at a coarser granularity, the request cannot be answered accurately and the institution must state its limitation. And where the source data has been overwritten by subsequent periods, the historical request cannot be answered at all. The third and fourth cases are prevented by a retention decision rather than by a capability decision, and the decision should be made when the landing zone is designed. The cost of retaining raw source data at full granularity is storage; the cost of not retaining it is the inability to answer a question about a past period, which cannot be remedied retrospectively [49], [52].

The Reporting Calendar as a Contested Resource

The reporting calendar is a shared resource and its contention is a design problem. Several obligations frequently fall due in the same week, each with its own source cut-off, control cycle and approval sequence. Where the platform's capacity was sized for the average demand, the peak demand determines whether deadlines are met, and the peak is where quality pressure is greatest and where the temptation to abbreviate a control is strongest. The architectural responses are three. Capacity is sized to the peak rather than the average, and the peak is computed from the calendar rather than assumed. Processing is scheduled so that obligations whose deadlines are furthest away are processed first, which spreads the load and leaves the tightest deadlines with the most remaining capacity. And the control cycle is designed to be completable within the window rather than to require a window that does not exist, which in practice means that the controls that must run on the critical path are identified and the remainder are run off it.

Projects and Extensibility of the Reporting Model

Where the reporting model is conformed, the extension required to serve a new obligation is usually small: a new projection, a new set of mappings, and any measures the new framework requires that the model does not already hold. Where the model is bespoke per obligation, the extension is a new instance of the whole model, with its own definitions and its own reconciliation burden. The difference is measurable in advance, and the measurement is worth making at design time: the number of governed artefacts that must be created to serve an additional obligation, given the obligations in the inventory. A conformed model with a complete mapping layer answers the question with a small number and a

bespoke model with a number proportional to the obligation's size. The same measurement answers the change question: how many artefacts must be modified when a shared definition changes. An enterprise architecture for transforming operational data into standardized regulatory information and reporting outputs. The Operational Data Sources include loan accounts, trading positions, collateral records, and legal entities. These sources supply account balances, trading positions, collateral values, and legal entity attributes to the Data Model Layers, which organize information into raw, standardized, conformed, and regulatory aggregation layers. This progression supports data cleansing, standardization, business-rule conformance, and the calculation of risk measures before the data is used for regulatory reporting.

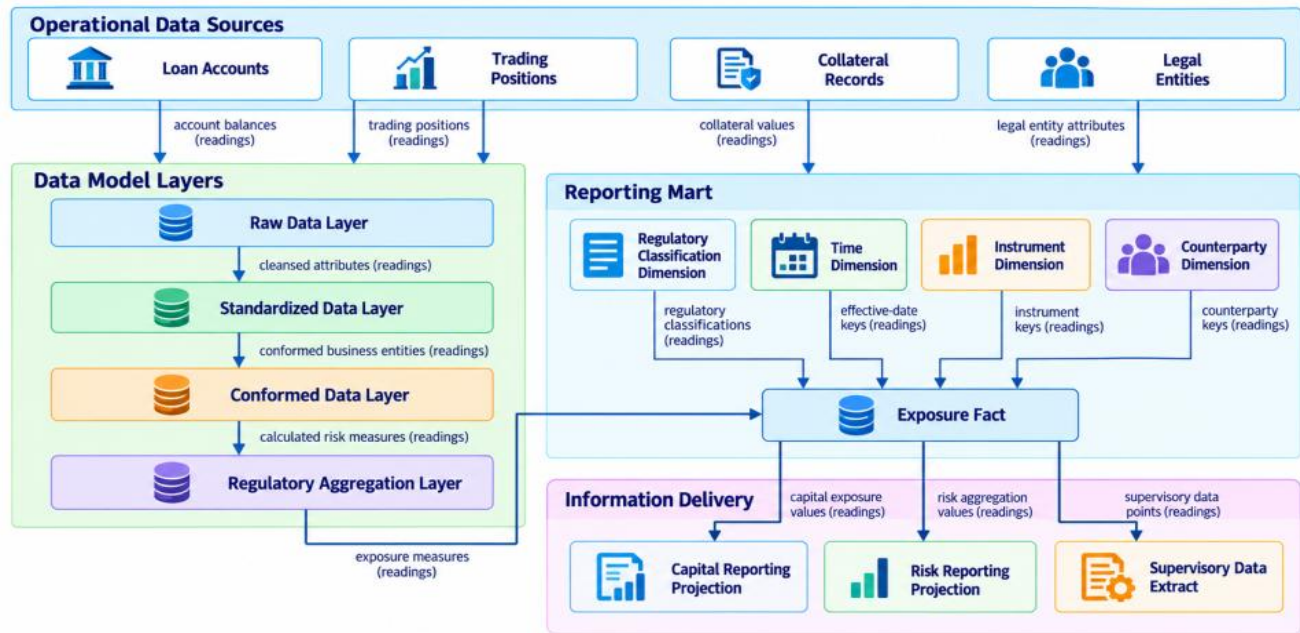


Figure 11.4 Enterprise Regulatory Data Modeling, Exposure Aggregation, and Reporting Architecture

The Reporting Mart organizes the information required for exposure analysis through regulatory classification, time, instrument, and counterparty dimensions. These dimensions provide classifications, effective-date keys, instrument keys, and counterparty keys to the Exposure Fact, which consolidates exposure-related measures for downstream consumption. This dimensional structure supports analysis across reporting periods, financial instruments, counterparties, and regulatory categories while maintaining a consistent representation of exposure information. The Information Delivery layer transforms the consolidated exposure data into outputs for capital reporting, risk reporting, and supervisory data extraction. The architecture establishes a flow from operational source systems through layered data processing and exposure aggregation to specialized reporting outputs. Overall, it illustrates how governed data models and a dimensional reporting mart can support consistent risk calculations, regulatory classifications, and multiple regulatory information delivery requirements.

CHAPTER 12

Regulatory Reporting Pipelines, Submission Controls, and Validation

The reporting data model determines what can be said. The reporting pipeline determines whether what is said is correct, complete, timely and defensible. This chapter addresses the controls applied between the aggregation layer and the submitted return, and the operational practices that make submission reliable. The chapter is organised around a claim that may appear obvious and is frequently violated: the control of a submission is a distinct engineering problem from the computation of the figures it contains. The computation is characterised by complex logic and moderate volume; the submission is characterised by simple logic, strict format requirements and an immovable deadline with a binary outcome. Separating the two allows each to be engineered for its own risk profile.

The Submission Pipeline

Figure Regulatory Submission Pipeline presents the submission pipeline as five stages: generate, validate, review, render and submit. Each stage has a distinct control responsibility, and the stages are presented in the order in which they must execute.

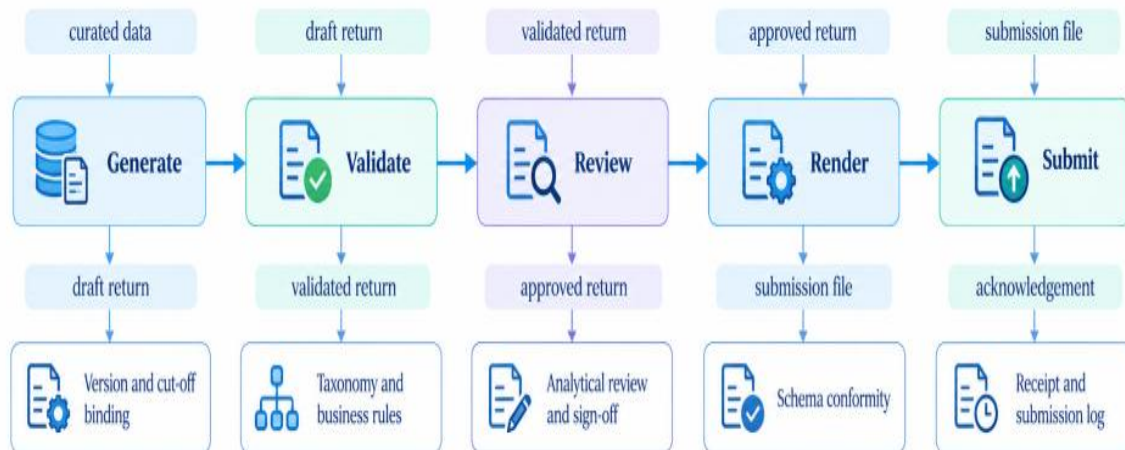


Figure 12.1 Regulatory Submission Pipeline

Generate

The generation stage produces the draft return from the curated data. Its principal controls are binding controls: the draft must be bound to a specific reporting period, a specific cut-off, a specific data version, a specific rule set version and a specific taxonomy release. Binding is a control because unbinding is a silent failure. A return generated from data extracted after the cut-off is not the return for the period. A return generated with reference data as it stands today is not the return as it should have been produced with the reference data in force. Neither condition produces a validation failure; both produce a materially different and potentially incorrect return. The binding should be recorded in the return artefact itself and in the metadata platform, so that the question what exactly was this return generated from? has a recorded answer rather than a reconstructed one.

Validate

The validation stage applies the checks described in Figure Validation Gates Before Submission and Table 12.1. Validation at this stage differs from the data quality controls of Chapters 4 and 8 in an important respect. Data quality controls assert expectations about the institution's data. Submission validation asserts expectations about the return as a

communication: that it conforms to the taxonomy, that its internal arithmetic is consistent, that its components sum to its totals, and that it is consistent with the institution's own prior submissions and with comparable disclosures.

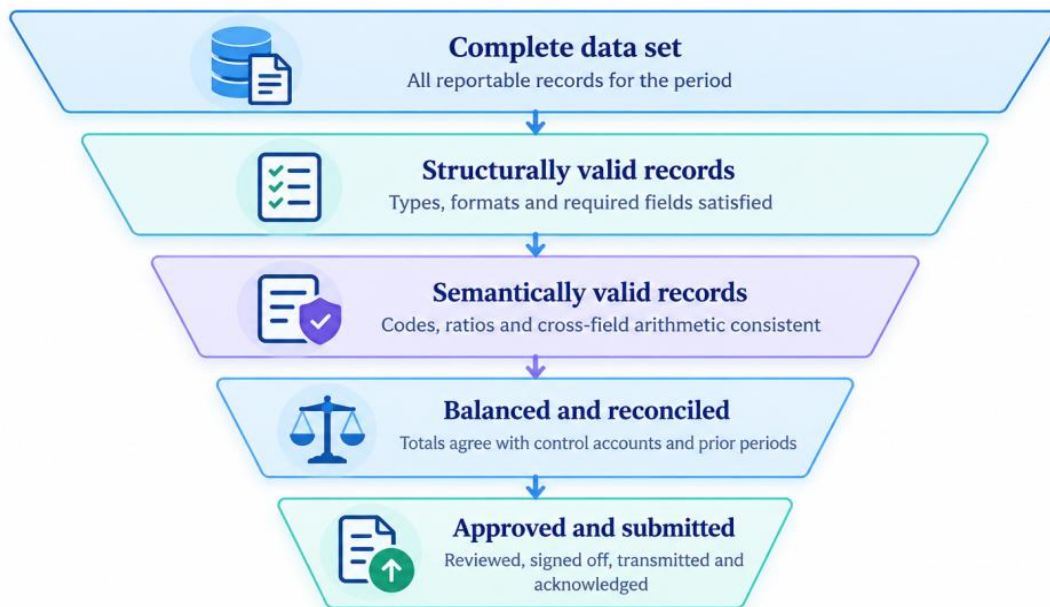


Figure 12.2 Validation Gates Before Submission

Review

The review stage is performed by people, and its purpose is to detect what automated validation cannot: a figure that is internally consistent and plausible in isolation but wrong in context. Analytical review is the technique. The return is compared against the previous period, against the same period in the prior year, against the institution's own published financial statements, and against the plan or the expectation of the business. Differences are investigated and explained.

The effectiveness of analytical review depends on its being structured rather than impressionistic. A review that asks a knowledgeable person to look at the return will produce variable results depending on their attention. A review that presents a defined set of comparisons with quantified movements and requires an explanation for each movement above a threshold produces consistent results and an auditable record.

Render

The rendering stage converts the reviewed return into the transmission format. Its controls are format conformance: that the file validates against the schema or specification the receiving authority requires, that identifiers resolve, and that the structure matches the taxonomy release. Format failures are the most common cause of submission rejection, and they are entirely avoidable through pre-submission validation against the published schema. Where the authority publishes a validation tool, it should be executed before submission. Where it does not, the institution should build an equivalent check.

Submit

The submission stage transmits the return and records the acknowledgement. Its controls are operational rather than analytical: that the transmission occurred within the deadline, that an acknowledgement was received, that the acknowledgement references the correct return, and that the transmission is recorded in an immutable log with a timestamp. The acknowledgement is evidence and should be retained with the return. Where the authority acknowledges receipt but later reports a validation error, the acknowledgement and the error response together form the evidence of what was transmitted.

Submission Control Checkpoints

Table 12.1 sets out the control checkpoints associated with the reporting cycle, together with the timing, the accountable role and the evidence produced.

Table 12.1 Submission control checkpoints with timing, accountability and evidence

Checkpoints	Control exercised	Timing	Evidence produced
Period definition	Confirmation of period boundaries and cut-off	Before data extraction	Period definition record
Data cut-off	Freezing of the extraction boundary for the period	At cut-off date	Cut-off certificate and extraction log
Completeness	Agreement of population against independent control totals	After loading, before aggregation	Completeness control results
Rule execution	Confirmation that all rules executed successfully	After transformation	Rule execution results with versions
Reconciliation	Agreement of aggregates with independent sources	Before return generation	Signed reconciliation statement
Return generation	Binding of the return to period, data and rule versions	At generation	Generation metadata record
Taxonomy validation	Conformance of the return to the published taxonomy	After generation	Validation report and error clearance
Arithmetic validation	Agreement of components, subtotals and totals	After generation	Internal consistency report
Analytical review	Explanation of all movements above defined thresholds	Before approval	Review record with explanations
Segregation of duties	Approval by an authority independent of preparation	Before submission	Approval record with role and timestamp
Submission	Transmission within deadline with acknowledgement	At submission	Acknowledgement and transmission log
Post-submission review	Assessment of errors, warnings and authority queries	After submission	Query log and resolution record
Restatement control	Governance of any correction to a submitted return	On correction	Restatement approval and rationale

The Review Threshold and Its Calibration

The analytical review threshold determines which movements are investigated and which are accepted. Its calibration has consequences on both sides. A threshold set too high allows material movements to pass unexplained. A threshold set too low produces an unmanageable volume of explanations, which are then produced perfunctorily and provide no assurance. The practical calibration begins from the institution's materiality framework, adjusted downward to account for the fact that individually immaterial movements may aggregate to a material total. Many institutions use a two-part threshold: an absolute amount above which every movement is investigated, and a smaller amount above which movements are investigated whenever the aggregate of unexplained movements approaches materiality. The threshold should be expressed in the unit of the reported quantity, which differs across data points. A single percentage threshold across all data points will be simultaneously too loose for small-balance data points and too tight for large ones. Data-point-specific thresholds derived from materiality are more effective and require more configuration.

Segregation of Duties in Submission

The approval of a regulatory return should be performed by someone who did not prepare it. In a small reporting function, this is achieved by having the preparer and the approver be different individuals, with the approver's review documented. The control that matters most in practice is the approver's ability to refuse. An approval process in which the approver lacks the standing, the information or the time to challenge the preparer is a formality. The architectural response is to give the approver access to the same evidence the preparer has: the reconciliation results, the validation results, the analytical review record, the exception dispositions and the open waiver list. An approver who can see the open waivers can form a view about whether the return is releasable.

Handling Rejection and Resubmission

Where a return is rejected by the authority, or accepted with warnings, the institution must correct and resubmit. The process requires the same discipline as the original submission, and it frequently does not receive it because it is performed under time pressure. Three requirements should be established in advance. The first is an identified authority empowered to approve a resubmission within the deadline, so that approval does not become the bottleneck. The second is a defined scope for the correction: whether the resubmission addresses only the reported error or whether the opportunity is taken to correct other known defects. The conservative and generally correct position is to correct the reported error and to record the other defects for the following cycle, because an expanded resubmission introduces new risk under time pressure. The third is a record of the difference between the original and the resubmitted return, retained as evidence of exactly what changed.

Restatement Governance

A restatement occurs where a return already submitted is corrected in a subsequent submission. Restatements are sometimes required by the authority, sometimes by the institution's own discovery of an error, and sometimes by a change in interpretation. Restatements are sensitive because they are visible. A pattern of restatements invites supervisory attention, and the institution should treat a restatement as an event requiring the same scrutiny applied to any other control failure. The governance requirements are that the restatement be approved by an identified authority, that its cause be classified, that its effect on every affected obligation be assessed rather than only on the obligation that triggered it, and that the classification feed into the root-cause analysis described in Chapter 10. The third requirement is the one most frequently missed: an error discovered in one return frequently affects others produced from the same data, and the assessment should be performed systematically rather than left to recollection.

Timeliness and the Reporting Calendar

Timeliness is a quality dimension discussed in Chapter 4, and at the reporting layer it becomes a scheduling discipline. The reporting calendar should be constructed backward from the submission deadline, allocating time to each stage: data availability, loading, validation, reconciliation, generation, review, approval and transmission. The allocation should include explicit contingency, because every stage is subject to variance and the deadline is not. The calendar should be maintained as an artefact and should be reconciled against actual performance after each cycle. The systematic comparison of planned against actual timings is the mechanism by which bottlenecks are identified. Without it, the institution knows that it submits on time but not where the time is consumed, and it cannot improve the process except by adding resource indiscriminately.

Contingency and Degraded Operation

There will be cycles in which a source is unavailable, a pipeline fails or a deadline is brought forward. The institution's ability to submit under degraded conditions depends on decisions taken in advance. Three provisions are useful. A defined minimum viable return identifies the subset of the return that cannot be omitted and the subset that can be completed later under a supervisory extension or a corrected resubmission. A manual fallback identifies, for each automated stage, whether a manual alternative exists and what it costs in control quality. An escalation path with defined triggers states the conditions under which the reporting function escalates to senior management and to the supervisor, so that escalation does not depend on an individual's judgement under pressure. The third provision is the one most frequently absent and the one that most reliably distinguishes a well-governed reporting function. An institution that has agreed in advance that it will notify its supervisor of a late return if a specified condition is met will notify. An institution that has not will improvise, and improvised notification frequently occurs after the deadline rather than before it.

Automation of the Submission Process

Automating submission converts a manual, individually supervised process into a scheduled one. The benefits are reduced effort, reduced manual error and a more consistent evidence trail. The risks are that a defect in the automation is reproduced in every cycle and that control is relaxed because the process appears automated. Three controls address the risks. The automation itself should be subject to the change control described in Chapter 8, with versioning and testing. The outputs should continue to attract analytical review, because automation removes the manual inspection through which defects were previously noticed. And the automation should be monitored for silent failure: a submission process that runs successfully but transmits an empty or truncated file must be detected, which requires a post-submission verification that the acknowledgement references a return of the expected size and content.

Worked Example: Reducing an Analytical Review Record

The analytical review record for a quarterly return has three components, and their design determines whether the review is effective or ceremonial. The first component is a comparison set, generated automatically: current period against prior period, current period against the same period last year, and the institution's own published financial figures against the corresponding return data points. The comparisons are produced as a table with absolute and percentage movements, so that a reviewer examines a uniform artefact rather than assembling the inputs. The second component is a threshold application, using materiality-derived thresholds that distinguish movements requiring explanation from those that do not. The thresholds are data-point-specific rather than uniform. The third component is an explanation record, which requires a written explanation for each movement above threshold and records the name of the person who provided it and the date. Where a movement is expected and recurring, the explanation is carried forward from the prior period with a confirmation rather than retyped, so that the reviewer's attention is directed to new movements. The record is retained with the return. Its value during an examination is that it demonstrates that the reported figures were reviewed by a person with the knowledge to assess them, on a defined basis, before submission.

The Control Environment Around Submission

Submission is the point at which the institution commits to a figure, and the controls around it are therefore about authority rather than about data. Four controls operate at this point. The completeness of the internal approvals asserts that every accountable owner has confirmed the figures within their domain. The formal release decision asserts that a named individual with the authority has released the return, and records the basis on which the release was made, including any outstanding exceptions and the reasons they do not preclude submission. The integrity of the transmitted artefact asserts that what the gateway received is what the institution produced, by checksum or by comparison of the submission acknowledgement against a locally computed digest. And the reconciliation of the acknowledgement to the return asserts that the gateway accepted the return as submitted rather than accepting it subject to warnings, and records any warning and its disposition. The second control is the one most often diffuse in practice. Where release authority is held collectively or by a team, no individual commits to the figure, and a subsequent question about why a return was submitted with a known defect has no accountable answer [54].

Managing Late Source Data

Late-arriving source data is normal in a reporting operation and is not, in itself, a defect. What constitutes a defect is late data entering a submitted period without the change being recognised as a revision. The control design distinguishes three cases. Data arriving before the cut-off is processed normally and requires no special treatment. Data arriving after the cut-off but before submission is processed through a controlled change in which the effect on the reported figures is quantified, the affected controls are re-executed and the change is recorded, so that the figures submitted are the latest available and their lateness is documented. Data arriving after submission is processed through the revision process, in which the institution determines whether the change is material, notifies the supervisor where the framework requires it, and records both the original and revised figures. The failure mode the design prevents is the third case being handled as though it were the second, which produces a reported figure that differs from the submitted figure with no record of why, and which is discovered at the next examination rather than at the time of the change [49], [57].

Period-End Discipline

Reporting quality is determined disproportionately by the discipline of the three days around the period end, and the discipline is a designed property rather than a matter of individual conscientiousness. Three practices constitute it. The cut-off is enforced technically, so that data entering after the cut-off is physically routed to the next period's landing area rather than being accepted into the closed period. The control closure is gated: the period cannot be released while any control of defined materiality is failing without a recorded and approved waiver. And the reprocessing window is bounded: the length of time for which the closed period may be reopened is defined by policy, and a reopening beyond that window requires authorisation at a defined level. Each practice converts an expectation about behaviour into a property of the system, which is the only form of control that operates reliably under the pressure of a deadline.

Restatement: When and How

A restatement is the resubmission of a previously reported figure. It is expensive, it draws attention, and it is occasionally the correct and required action. The decision framework should be established before the situation arises. Three questions determine the response. Is the corrected figure materially different from the submitted figure, judged against the materiality basis the institution applies to that obligation? Is the error a violation of the framework's requirement as opposed to a refinement of an estimate that the framework permits to vary? And does the framework impose a positive obligation to notify, or does it require resubmission only on request? Where the answer to the first two questions is yes and the third is a notification obligation, the institution notifies. Where the corrected figure is immaterial or the change is a permitted refinement, the institution records the difference internally and carries the correction into the next period. In every case the institution records what it knew, when it knew it and what it decided, because the record of the decision is what distinguishes a considered judgement from an omission.

Automation of the Submission Path

Automating submission is frequently the last step taken and the one that yields the largest reduction in operational risk, because the manual steps around submission are concentrated at the point of highest consequence. Three components are candidates for automation and each removes a class of error. The rendering of the return from the governed figures removes transcription error. The structural validation of the rendered artefact against the framework's schema removes format rejection. And the transmission with acknowledgement capture removes the ambiguity about whether a submission occurred and what was accepted. What should not be automated is the release decision. The decision that the figures may be submitted is an accountable judgement, and an automated release removes the accountability without removing the need for the judgement. The design that satisfies both is an automated path from governed figures to a validated artefact, with a single manual authorisation at the point of transmission. Figure Submission Control Checkpoints lists the checkpoints through which a submission passes and the evidence each checkpoint produces, from the internal approvals to the reconciliation of the gateway acknowledgement.



Figure 12.3 Submission Control Checkpoints

The figure illustrates an end-to-end architecture for preparing, validating, approving, and submitting regulatory returns. The Return Preparation stage uses the reporting data mart as its source and passes calculated reporting values to the return generation engine. Period and version binding associates the generated return with the appropriate reporting period and version, supporting controlled preparation and traceability of reporting outputs.

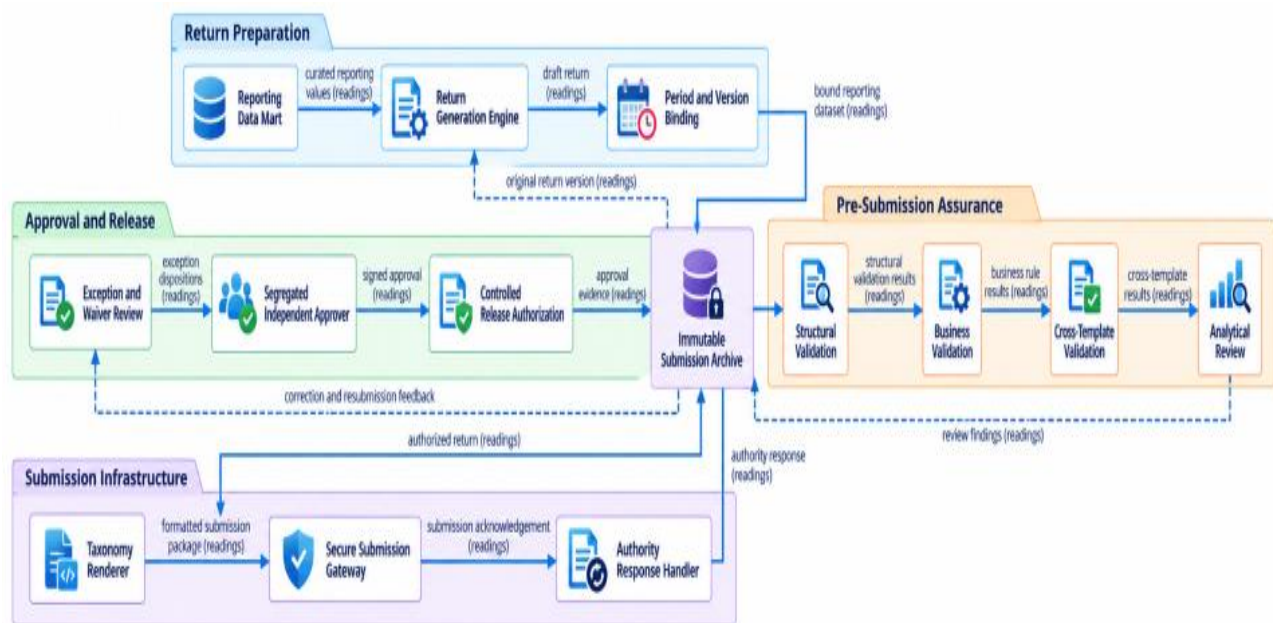


Figure 12.4 Regulatory Reporting Preparation, Pre-Submission Assurance, and Controlled Filing Architecture

The Pre-Submission Assurance stage applies structural validation, business validation, and cross-template validation to check the completeness and consistency of the prepared return. The validated output proceeds to analytical review, where reporting results can be examined before submission. In parallel, the Approval and Release process manages exceptions, segregates independent approval responsibilities, and performs controlled release authorization. An immutable submission archive preserves the authorized return and provides a record of the version approved for submission. The Submission Infrastructure layer includes a taxonomy renderer, secure submission gateway, and authority response handler. These components support formatting the return for submission, securely transmitting the filing, and processing acknowledgments or responses from the relevant authority. Feedback from validation, analytical review, and submission responses can support correction and resubmission where required. Overall, the architecture combines validation, approval controls, immutable archiving, secure delivery, and authority-response handling to strengthen the reliability and auditability of regulatory submissions.

CHAPTER 13

Data Governance, Security, Privacy, and Regulatory Accountability

The technical architecture described in the preceding chapters produces correct reporting only if decisions about data are made deliberately, by identifiable people, following documented processes. That is the substance of governance, and this chapter treats it as an architectural concern rather than as an organisational overlay. It then addresses the security and privacy requirements that constrain the architecture, and the accountability arrangements that make the whole system defensible.

Governance as Decision Rights

Figure Governance Operating Model presents the governance operating model as five lanes with the decisions taken at each. The model's organising principle is that governance consists of decision rights and accountability rather than of committees.



Figure 13.1 Governance Operating Model

The literature on data governance makes this point explicitly, framing governance in terms of which decisions must be made, who makes them and who is accountable for the outcomes [36]. That framing is more useful than a committee structure because it forces the institution to state its decisions rather than its meetings.

The Decisions That Must Be Made

Table 13.1 sets out the decision domains that a regulatory data governance model must address, with the accountable role and the primary artefact for each.

Table 13.1 Governance Decision Domains with Accountable Roles and Primary Artefacts

Decision domain	Decision to be made	Accountable role	Primary artefact
Domain definition and ownership	Which data domains exist and who owns each	Executive data council	Domain ownership register
Semantic definition	What each reported concept means precisely	Data owner with reporting function	Business glossary entry
Classification standards	Which codes and taxonomies are adopted and how mapped	Reference data owner	Reference data mapping and policy
Source of record designation	Which system is authoritative for each attribute	Data owner	Source-of-record register
Quality thresholds	What level of defect is acceptable for each dimension	Data owner with risk function	Threshold register
Exception disposition	Whether a defect is remediated or waived	Data steward with control function	Exception and waiver records
Rule change approval	Whether a transformation change may be released	Rule owner with change authority	Rule inventory and approval record
Access entitlement	Who may access which data and for what purpose	Data owner with security function	Entitlement register and access log
Retention and disposal	How long data and evidence are retained	Records management owner	Retention schedule
Reporting release	Whether a return may be submitted	Reporting accountable executive	Approval record
Restatement approval	Whether a submitted return may be corrected	Reporting accountable executive	Restatement approval record

Three properties of this table deserve comment.

First, several decisions are jointly owned, and the joint ownership should be specified rather than left to negotiation. The quality threshold decision, for example, requires the data owner to state what is achievable and the risk function to state what is acceptable, and the decision is the reconciliation of the two. Where the decision right is unclear, the threshold will be set by whichever party is more assertive, which is not a governance outcome. Second, every decision domain has an artefact. A decision that is not recorded is not a governance decision but a conversation. The artefacts are the evidence that governance operated. Third, the decisions are few. A governance model with three hundred decision types is not a governance model. The value of the list is that it is short enough to assign and to audit.

The Accountability Matrix

Figure Accountability Matrix for Regulatory Data presents the accountability matrix for regulatory data. Its purpose is to make the assignment of responsibility and consultation explicit for each activity in the data lifecycle. The matrix distinguishes the accountable role, which is the single role answerable for the outcome, from consulted roles, which contribute to the decision. The distinction matters because ambiguity about who is accountable is the most common failure of data governance. Where two roles believe they are accountable, decisions are contested; where none does, decisions are not made. The matrix should be maintained as an artefact and reviewed when roles change. A matrix that reflects a previous organisational structure is worse than no matrix, because it misdirects escalation.



Figure 13.2 Accountability Matrix for Regulatory Data

The Data Owner and the Data Steward

Two roles are central to the model and are frequently confused.

- The data owner is accountable for a domain: for its definitions, its quality, its controls and its access entitlements. The role is a business role, not a technical one, because the decisions it makes are about meaning and acceptability rather than about implementation. A data owner who cannot decide what a term means in their domain is not functioning as a data owner.
- The data steward performs the day-to-day work on behalf of the owner: resolving exceptions, maintaining definitions, assessing the impact of proposed reference data changes, and reviewing quality results. The steward role requires domain knowledge and sufficient authority to make routine decisions without escalation.

A governance model that assigns ownership but not stewardship produces owners who are accountable for work they have no capacity to perform. A model that assigns stewardship without ownership produces stewards who cannot resolve the disputes that their work generates. Both roles are required, and their boundary should be documented.

Governance of the Metadata and Lineage Capability

The metadata and lineage capabilities described in Chapter 6 are themselves governed assets, and their governance is frequently neglected because they are regarded as tooling. Two decisions require ownership. The first is the standard for business metadata: what fields every catalogued asset must have, who is responsible for populating them and what the quality threshold is. Without a standard, the catalogue's completeness depends on individual diligence. The second is the lineage coverage target and its exceptions: what proportion of reported data points must have complete column-level lineage, and how exceptions are approved. Without a target, coverage drifts downward and the gap is discovered during an examination.

Security Architecture

Figure Security and Access Control Architecture presents the security architecture as a layered stack from identity through monitoring. The treatment here is confined to the aspects that are specific to a regulatory data platform; general information security practice is outside the scope of this book.

Identity and Entitlement

Access to regulatory data should be granted against an identity that is attributable to a natural person or to a specifically identified service, and the entitlement should be recorded with the reason for its grant. The design point that matters most in a regulatory context is that entitlements must be reconstructible for historical periods. An examination may ask who had access to a dataset at a past date, and answering requires the entitlement history rather than the current state. Where entitlements are held only in their current form, the institution cannot answer. Service identities require explicit

governance. A pipeline that accesses data using a shared credential cannot be attributed, and the credential's access cannot be reduced without affecting every pipeline that uses it, which discourages least-privilege design. Each pipeline should have its own identity with access limited to what it reads and writes. The additional administrative burden is repaid in the ability to answer the attribution question.

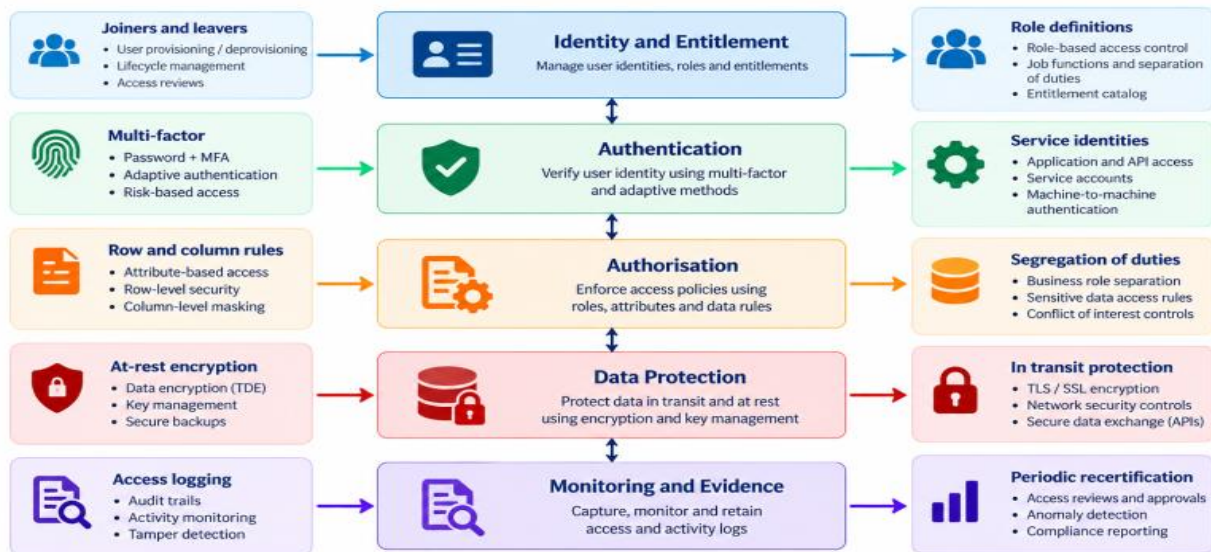


Figure 13.3 Security and Access Control Architecture

Authorisation for Regulatory Data

Authorisation for regulatory data has three requirements that distinguish it from authorisation for operational data.

The first is segregation of duties. The person who can modify a transformation rule should not be the person who can release a return. The person who can approve a waiver should not be the person who benefits from it. These separations should be enforced technically wherever the platform supports it, and by documented compensating controls where it does not.

The second is purpose limitation. Regulatory data is frequently also personal data, and its use should be limited to the purpose for which it was collected. Technically, this is expressed as domain-based access in which datasets are grouped by purpose and entitlements are granted at the level of the domain.

The third is row and column level control where the data is granular enough to require it. Where a reporting dataset contains data attributable to individual customers or employees, access should be controllable at the level of the attribute rather than only at the level of the dataset.

Encryption and Key Management

Data should be protected at rest and in transit using standard mechanisms, and the treatment here is confined to two considerations specific to regulatory platforms. The first is that archival data must remain readable for the retention period. Encryption is only as durable as the key management. An archive encrypted with a key that cannot be recovered after a personnel change or a platform migration is not an archive. Key management should be designed for the retention period rather than for the operational period. The second is that reprocessing requires access to historical data. Where data is encrypted and the keys are rotated, the older data must remain accessible for reprocessing. The rotation policy should be built around the reprocessing requirement rather than imposed independently of it.

Monitoring and Access Evidence

Access to regulatory data should be logged, and the logs should be retained for the period required by the institution's policies. The receipt of access logs is that they allow a question about who accessed a dataset in a past period to be answered, which is a recurrent examination question. Periodic recertification the process by which data owners confirm

that existing entitlements remain appropriate is the mechanism that prevents entitlement accumulation. It is frequently performed as a compliance exercise in which all entitlements are confirmed without scrutiny. Its value depends entirely on whether it involves actual review, and a recertification that never reduces an entitlement should be treated as a control failure rather than as evidence of control.

Privacy and Data Protection

Where regulatory data includes personal data, data protection requirements constrain the architecture. The most directly relevant framework is the General Data Protection Regulation in the European Union, which imposes obligations on lawfulness, purpose limitation, data minimisation, accuracy, storage limitation, integrity and accountability [55].

Reconciling Reporting Obligations with Data Protection

The reconciliation is not a conflict but requires explicit reasoning. Regulatory reporting obligations are typically grounded in a legal obligation or in a public interest, which provides a lawful basis for the processing. The data protection requirements then constrain how the processing is performed rather than whether it may occur.

Four constraints are most consequential architecturally.

- **Data minimisation** requires that the data collected and retained be adequate and relevant to the purpose. In a reporting context this conflicts with the reproducibility requirement and with the granularity principle of Chapter 11, both of which argue for retaining more data for longer. The practical resolution is to retain reportable data for the retention period required by the reporting obligation, and to justify the retention explicitly against the reporting purpose rather than to retain everything indefinitely by default.
- **Storage limitation** requires that personal data be kept no longer than necessary. In a reporting context, the necessity is determined by the reporting and records obligations. The retention schedule should be derived from those obligations and documented, rather than set at a round number of years.
- **Accuracy** in data protection terms requires that personal data be accurate and, where necessary, kept up to date. This aligns closely with the data quality requirements of Chapter 4, and the controls are largely shared. Where they diverge is in the treatment of historical reporting data, which must reflect the state as at the reporting date even if it has since been corrected at source.
- **Accountability** requires the institution to demonstrate compliance. In a data platform context this means that the lawful basis for each processing activity, the retention period and the access controls must be recorded and available, which is a metadata requirement and reinforces the treatment in Chapter 6.

Anonymisation and Its Limits

Where reporting can be satisfied without personal data, anonymisation reduces risk. Its limits in a reporting context should be understood.

Regulatory reporting frequently requires aggregation to a level at which individuals are not identifiable, which is a form of anonymisation by aggregation. The strength of the protection depends on the granularity of the aggregation and on the presence of other data that could be combined with it. A dataset containing exposures by counterparty at a fine granularity may permit re-identification even without names, because the granularity itself is identifying.

Where a stronger protection is required, techniques exist that provide formal privacy guarantees, of which differential privacy is the best known [62]. Their application to regulatory reporting is constrained by the requirement that reported figures be exactly reproducible and directly reconcilable to source records, which formal privacy mechanisms cannot provide by construction. The practical position is that these techniques are relevant to analytical and derived disclosure use cases rather than to the reported figures themselves.

Special Categories and Cross-Border Considerations

Where data includes special categories of personal data or is subject to transfer restrictions, the architecture must accommodate the additional requirements. The design consequence is normally that the affected data is separated at the ingestion layer so that its handling can be controlled distinctly.

Accountability and Assurance

Governance produces decisions; assurance establishes whether the decisions were correct and whether they were followed.

The Three Lines

The three-lines model of assurance is widely used in financial institutions and maps naturally onto a data governance framework.

The first line is the operational function that produces and controls the data: the data engineering and reporting teams. Its responsibilities include the design and operation of controls, the remediation of exceptions and the maintenance of evidence. The second line is the risk and control function that establishes the framework, monitors its operation and challenges the first line. In a data governance context, the second line defines the control standards, reviews the control results and tests the design of controls.

The third line is internal audit, which provides independent assurance on the design and operation of the control framework. The model is useful provided the distinction is real. Where the second line performs the first line's work, the challenge function is lost. Where internal audit performs testing that the second line should perform, resources are consumed without improving the framework.

The Evidence Book

The practical artefact that makes assurance efficient is a maintained evidence book for the reporting platform: an organised, indexed collection of the evidence that controls operated, for each period. Its contents include the control execution results, the reconciliation statements, the exception and waiver records, the approval records and the submission acknowledgements. The evidence book's value is that it converts an examination into a retrieval exercise. Where the evidence exists but is scattered across systems and teams, the examination consumes weeks of skilled effort in gathering rather than in explaining.

Independent Model and Data Validation

Where reported figures depend on models, the model risk management expectations discussed in Chapter 2 apply. The relevance to this chapter is that the validation of a model includes the validation of its data inputs: their completeness, their timeliness and the correctness of their transformation. A model validated only on its mathematics and not on its inputs has been only partially validated.

Worked Example: Making the Access Question Answerable

A supervisor asks which individuals had access to the counterparty exposure dataset at a date eight months past. Three institutional arrangements produce three different answers. Under the first, access is held in a group's membership in the platform's current directory. The question cannot be answered for a past date, because membership history is not retained. The institution can state who has access now. Under the second, entitlement grant and revocation events are logged to a security event store with retention sufficient to cover the examination window. The question is answered by reconstructing the entitlement set as at the date. The answer is accurate and it takes an analyst an hour to produce. Under the third, entitlements are held as versioned records in the same catalogue that holds the data's other metadata, so the question is a direct query against a governed asset. The answer takes minutes and is reproducible by anyone with catalogue access. The arrangement chosen determines not merely the effort of answering but the institution's ability to answer at all. Where entitlement history is not retained, the honest answer to the supervisor's question is that the institution cannot state the position at the past date, which is a materially worse answer than an accurate one.

Accountability for Automated Controls

Automated controls raise a governance question that manual controls do not: who is accountable for a control that no person executes? The answer this book adopts is that accountability rests with the control's designer and owner rather than with its operator. The rule's accountable owner is responsible for whether the expectation it asserts is correct, whether its threshold is defensible and whether its population is the right one. The operations function is responsible for whether it executed. The distinction matters because it prevents the failure mode in which an automated control is presumed correct because it runs reliably, when in fact reliability of execution says nothing about correctness of design.

Segregation of Duties in a Small Reporting Function

The segregation-of-duties principles described in this chapter are straightforward in a large function and difficult in a small one, and the difficulty should be resolved by design rather than by exception. Four incompatibilities matter most: the preparation of figures and their approval, the authoring of a rule and its independent review, the administration of access and the review of that access, and the execution of a control and the authorisation of a waiver from its result. Where the function is too small to separate by person, three approaches preserve the substance of the control. The separation may be temporal, with the same individual performing incompatible acts at different times and the sequence recorded. It may be evidential, with the second act being an independent verification whose result is retained, even where the verifier is within the same function. Or it may be escalated, with the approval moved to a function outside the reporting team. The second approach is the most commonly appropriate, and its effectiveness depends entirely on whether the verification is substantive rather than confirmatory, which is a question of the evidence it produces [54].

The Governance Artefacts That Actually Constrain

Governance frameworks produce many artefacts and few of them constrain behaviour. The artefacts that constrain are those referenced by the operating process. A policy constrains when a control's design is required to cite it. A standard constrains when a validation rule checks against it. A role definition constrains when the approval workflow will not complete without the named role. A decision right constrains when the change process routes a proposal to its holder and records the decision. A term definition constrains when mappings and controls use the term and are tested against its meaning. The test to apply to a governance artefact is therefore simple: identify the process step that cannot complete without it. An artefact for which that step cannot be named is documentation, and documentation has value but it does not constrain. Institutions that distinguish the two spend their governance effort where it operates.

CHAPTER 14

Scalable, Resilient, and Observable Regulatory Data Platforms

A regulatory data platform must operate reliably at the volume, latency and availability that its obligations require, and must make its own condition observable to the people accountable for it. This chapter addresses scalability, resilience and observability as properties of the platform rather than as separate concerns, on the argument that they share a common foundation: the platform must know its own state and be able to act on that knowledge.

The Reference Platform Architecture

Figure Reference Architecture for Resilient Regulatory Data Platforms presents a reference architecture consisting of processing and storage layers with cross-cutting concerns. The architecture is presented as a set of responsibilities rather than as a technology recommendation, and the technology choices within each layer are discussed below.



Figure 14.1 Reference Architecture for Resilient Regulatory Data Platforms

Storage Choices and Their Consequences

Regulatory data platforms use three storage families, with distinct properties. Relational stores provide transactional integrity, declarative query capability and mature access control. They are the natural home for conformed reference data and for the curated data structures that require referential integrity. Their limitation is scale: a relational store holding many years of transaction-level detail will encounter performance constraints that are difficult and expensive to resolve. Object stores provide effectively unlimited capacity at low cost with simple consistency semantics. They are the natural home for the immutable landing zone and for archived data. Their limitation is that they provide no query capability of their own; query must be provided by a compute engine that reads them. Columnar analytical stores provide efficient aggregate query over large volumes. They are the natural home for the reporting data marts and the aggregation layer. Their limitation is that they are optimised for reading aggregates and are less suitable for the transactional updates that reference data requires.

The architectural pattern that follows is the separation of the immutable landing zone in object storage from the curated and reference structures in relational stores and from the aggregation layer in a columnar store. The pattern is not novel; it is the practical convergence of cost, capability and consistency requirements.

Compute Choices

Batch engines process bounded datasets and are the backbone of periodic reporting, where the set-based semantics discussed in Chapter 9 are naturally expressible. Stream engines process unbounded data and are required where latency is a functional requirement. Rule engines interpret declarative specifications and are required where the metadata-driven approach of Chapter 8 is adopted. Report engines render and transmit. The significant architectural question is not which engines to use but how many. Each additional engine introduces its own programming model, operational tooling, security integration and skill requirement. An institution operating six engines pays six times the operational overhead, and the overhead is borne by scarce specialist staff. The practical guidance is to standardise on the smallest number of engines that satisfies the obligation profile, and to accept a degree of technical inelegance in exchange for operational simplicity. Distributed processing frameworks that provide both batch and structured query capability over shared storage have made this standardisation substantially more achievable [25].

Cross-Cutting Concerns

Orchestration, scheduling, secret management and configuration management are cross-cutting and should be provided as shared services rather than built separately by each pipeline. The reason is that their failure modes are systemic: a scheduling defect affects every pipeline, and a configuration inconsistency produces behaviour that varies by pipeline in ways that are extremely difficult to diagnose. The specific requirement that regulatory data imposes on orchestration is dependency-aware execution with recorded provenance: the orchestration metadata should record which version of which job executed with which parameters, because that record is part of the evidence base discussed in Chapter 13.

Scale and Its Limits

Scalability is the platform's ability to accommodate growth in volume, in the number of obligations and in the number of consumers.

Volume Scalability

Volume growth in regulatory reporting arises from finer granularity, longer history and additional obligations. The architectural responses are partitioning, which limits the scope of any operation, and the separation of storage from compute, which allows each to scale independently. The property that matters most in a regulatory context is that reprocessing remains feasible. A platform that can process the current period in an hour but cannot reprocess five years of history in any reasonable time is not scalable for regulatory purposes, because reprocessing is a routine operational requirement rather than an exception. Reprocessing capacity should be measured explicitly and treated as a requirement.

Change Scalability

The scalability that distinguishes a durable regulatory platform is not volume scalability but change scalability: the ability to accommodate a new obligation, a new source or a changed framework without redesign. The architectural properties that produce change scalability are those argued throughout this book: separation of the presentation layer from derivation, declarative rules with effective dating, conformed reference data with governed change, captured lineage and a canonical integration model. Each of these imposes cost at design time and returns it at the first significant change.

Organisational Scalability

A third dimension of scale is organisational. A central platform team that must review every change becomes a bottleneck as the number of pipelines grows. The architectural response, discussed in Chapter 2 as the federated pattern, is to distribute ownership while retaining central standards and a central conformance layer. The governance mechanisms described in Chapter 13 are what make the distribution safe, and their absence is what makes centralisation necessary.

Resilience

Figure Observability Signal Inventory compares recovery-oriented and continuity-oriented resilience strategies.

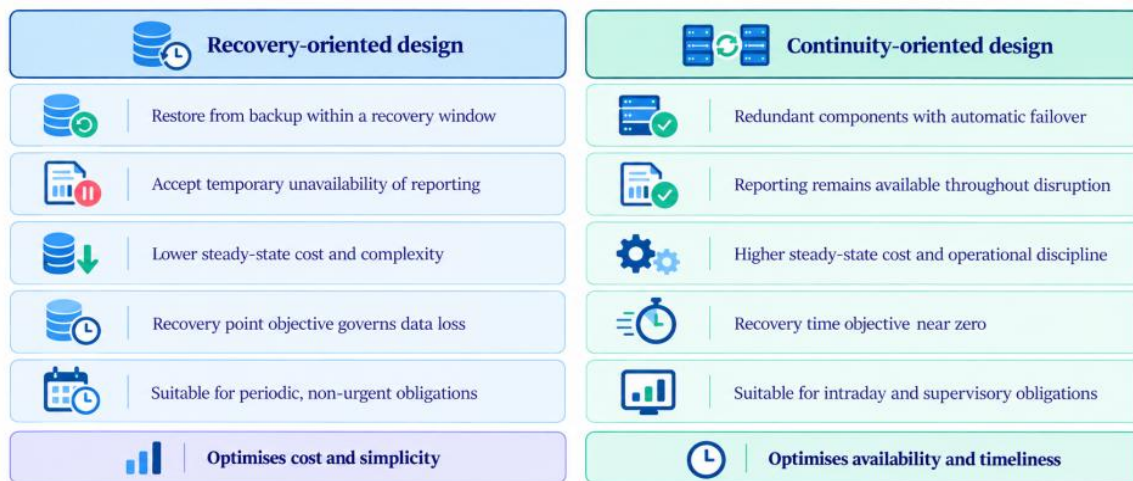


Figure 14.2 Resilience Strategies for Critical Data Services

Defining the Objectives

Resilience design begins with objectives expressed in the terms the reporting obligation uses. The recovery point objective states how much data the institution can afford to lose, expressed as a time. The recovery time objective states how long the platform may be unavailable. A maximum tolerable downtime, where the obligation cannot tolerate any interruption, is a special case of the recovery time objective approaching zero.

The objectives should be set per obligation rather than per platform, because the obligations differ. An intraday exposure monitoring capability has different objectives from a quarterly prudential return. Setting a single platform-wide objective produces either unnecessary cost or inadequate protection.

Redundancy, Failover and Data Replication

Where the objective requires continuity, redundancy is provided at each layer: multiple application instances, replicated storage and redundant network paths. Failover may be automatic or manual, and the choice is a function of how quickly detection and decision can occur.

Data replication raises a consistency question that is genuinely difficult. Synchronous replication guarantees that no committed data is lost and imposes latency on every write. Asynchronous replication imposes no latency and permits data loss bounded by the replication lag. The trade-off between consistency and availability under partition, articulated in the distributed systems literature, is inescapable [26], and the practical resolution is deliberate eventual consistency with bounded lag [25]. For regulatory reporting, the acceptable lag is derived from the recovery point objective, and the lag should be monitored because an unnoticed increase in replication lag silently increases the data at risk.

Testing Resilience

Resilience that has not been tested is a claim rather than a property. The testing obligation includes failover exercises, restore exercises and recovery-time measurement. The common failure mode is that restore exercises are performed against a subset of the data or in an environment that does not reflect production, producing a measured recovery time that does not apply. A restore exercise that measures the time to restore a single database is not evidence that the platform can be recovered within its objective. The exercise should restore the platform to the state required for reporting and should measure the time to that state, including the time to re-establish orchestration, reference data and entitlements.

Archival and Reprocessing

Archival serves two purposes: satisfying retention obligations and enabling reprocessing. The two require different properties, and conflating them produces an archive that satisfies neither. For retention, the requirement is durability, readability and completeness over a long period. For reprocessing, the requirement is that the archived data be usable as pipeline input, which means it must be accompanied by the code, rule versions, reference data versions and configuration

required to process it. An archive of data without its processing context cannot support reproduction, and the requirement to archive the context is the one most frequently overlooked.

Observability

Observability is the platform's ability to answer questions about its own state without having anticipated the specific question. It is distinguished from monitoring, which answers anticipated questions through predefined metrics.

The Signal Inventory

Table 14.1 sets out the observability signals that a regulatory data platform should expose, with what each indicates and the appropriate response.

Table 14.1 Observability Signals with Indication and Response Ownership

Signal class	What it indicates	Threshold basis	Response owner
Freshness	Lateness of data arrival relative to expectation	Arrival schedule with defined tolerance	Pipeline owner
Volume	Deviation of delivered volume from expected range	Historical distribution with statistical bounds	Pipeline owner
Completeness	Gap between delivered population and independent expectation	Control-total agreement or zero tolerance	Data steward
Schema conformance	Structural change in a source delivery	Schema contract validation outcome	Source owner with integration team
Quality score	Aggregate conformance against validation rules	Dimension thresholds by data point	Data owner
Reconciliation status	Agreement of aggregates with independent bases	Reconciliation tolerance by control	Reporting function
Reference data currency	Lag between reference publication and consumption	Defined adoption deadline	Reference data owner
Lineage coverage	Proportion of reported data points with complete lineage	Coverage target with approved exceptions	Metadata owner
Access anomaly	Unusual access pattern or failed authorisation	Baseline with deviation detection	Security function
Cost and resource	Consumption relative to provisioned capacity	Capacity planning thresholds	Platform owner
Job success and duration	Execution outcome and performance drift	Service level by criticality	Operations
Exception ageing	Accumulation of unresolved exceptions	Ageing thresholds by classification	Data steward with control function

From Signals to Action

Signals without responses produce dashboards that are not read. Three design principles convert them into an operating capability.

1. **Each signal has a named owner and a defined response.** The response states what the owner does when the signal breaches its threshold, not merely that they are informed. A signal whose response is review is a signal that will be reviewed once and then ignored.

2. **Thresholds are derived rather than observed.** The principle stated in Chapter 4 for validation rules applies equally to observability signals, and its violation produces the same result: thresholds that never trigger because they were set at the recent worst case.
3. **Signals are tiered by consequence.** Not every breach requires immediate action. Tiering allows the operations function to prioritise and prevents the escalation of every deviation to senior management, which would rapidly devalue escalation.

Observability of the Control Framework

The most valuable application of observability in a regulatory data platform is to the control framework itself. A dashboard that shows, for each reporting obligation, the completeness of its lineage, the currency of its reference data, the status of its reconciliations, the open exceptions and the age of the oldest unresolved item provides a direct view of the institution's readiness to report correctly. That view is useful to the reporting function continuously and is invaluable at the point when a supervisor asks a question about the institution's control environment.

Operational Practice

Three operational practices sustain the platform.

- **Run books** document, for each critical process, what it does, what its dependencies are, how to detect failure, how to recover and who to escalate to. Their value is greatest for processes that fail rarely, because those are the processes for which no one retains the knowledge.
- **Change management** ensures that every change to the platform follows a review and approval path proportionate to its risk. The specific requirement for a regulatory platform is that changes affecting reported figures be identified and subjected to the impact assessment described in Chapter 6. A change management process that cannot distinguish a cosmetic change from one that alters a reported figure provides no control where control is needed.
- **Post-incident review** converts failures into improvements. The review should establish the sequence of events, the control that should have detected the failure and did not, and the change that would prevent recurrence. A review that concludes with operator error has concluded nothing, because operator error is a design condition that the platform must accommodate rather than a cause that can be eliminated.

Capacity Planning for Reprocessing

Reprocessing is a routine requirement rather than an exception, and its capacity should be planned explicitly. The planning calculation begins with the largest plausible reprocessing demand. Four events generate demand: a historical period must be reproduced for an examination; a correction to a source extract requires reloading and downstream processing; a rule change requires re-derivation across the affected history; a platform migration requires rebuild of the reporting layer from the landing zone. The largest of these determines the requirement. The requirement is then expressed as a duration and checked against the reporting calendar. Where a full five-year reprocessing would consume three weeks, the institution must either accept that a reprocessing request during a reporting cycle cannot be met, or invest in capacity, or archive at a coarser granularity that reduces the volume to be reprocessed. Each is a legitimate choice; arriving at it by discovery during an examination is not.

The Cost of Observability

Observability has a cost and its allocation should be deliberate. Instrumentation adds processing overhead. Retaining signal history adds storage. Alerting adds operational attention, and operational attention is the scarcest resource in the system. The practical discipline is to treat each signal as an investment requiring justification: what class of failure does it detect, what is the cost of that failure going undetected, and what is the operational cost of the signal? Signals that cannot answer the first question should be removed, because unexplained dashboards teach operators to disregard dashboards.

Recovering a Failed Reporting Cycle

The recovery procedure for a failed reporting cycle is best designed as a sequence that has been rehearsed rather than assembled under time pressure.

The first step establishes the extent: which populations are affected, which obligations consume them, which controls have completed and which have not. Establishing extent before taking action prevents the common error of recovering more than was damaged, which consumes the window and introduces new risk.

The second step stops the propagation: affected batches are held, downstream consumers are notified and the cut-off for the affected period is suspended pending a decision. The third step establishes the decision point: whether the cycle can be completed within the deadline with a reduced control set, or whether the deadline requires notification. This decision belongs to the accountable owner rather than to the operations team, and it should be made once, with the options stated.

The fourth step executes the recovery, using the idempotent reload described earlier so that partial completion can be retried without investigation. The fifth step re-establishes and evidences: the controls that were completed before the failure are re-executed, since their results may have been invalidated by the state from which they ran, and the whole sequence is recorded as evidence. The rehearsal matters because the third step is the one that is hardest under pressure and the easiest to prepare, since its decision criteria can be written down, reviewed and agreed before it is needed.

Figure Resilience Strategies for Critical Data Services inventories the observability signals available to the platform and names the owner responsible for responding to each, distinguishing data signals from process and control signals.

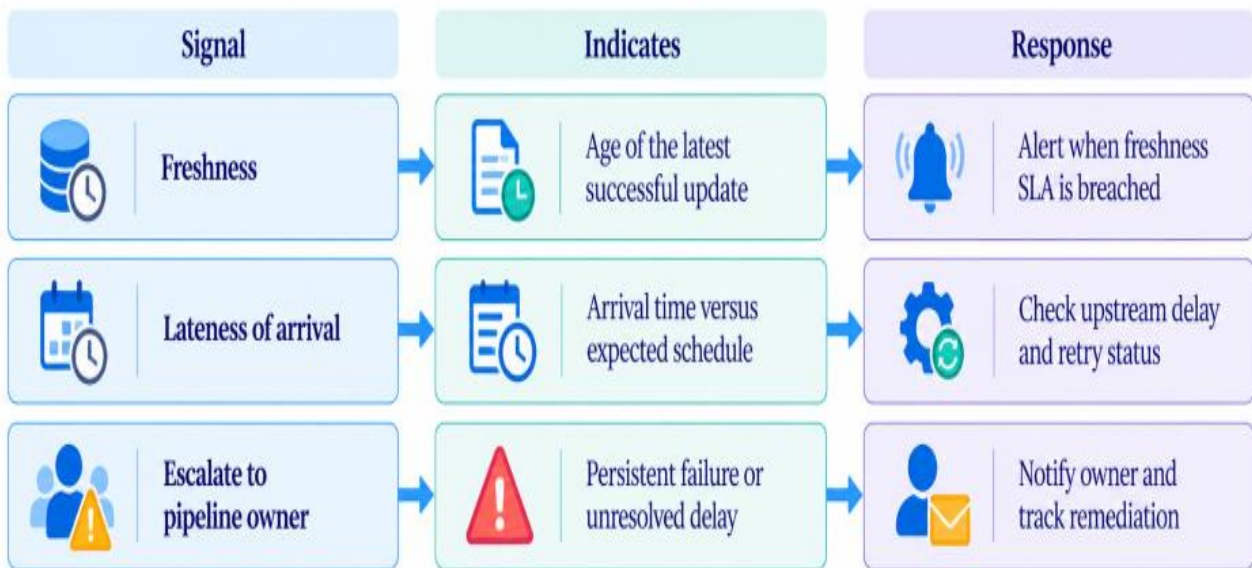


Figure 14.3 Observability Signal Inventory

CHAPTER 15

Enterprise Implementation Methodologies, Testing, and Operational Readiness

The preceding chapters have described what a regulatory data platform should be. This chapter addresses how it is built and, equally importantly, how it is shown to be built correctly. It presents a phased implementation methodology with defined deliverables and acceptance criteria, a layered testing strategy, and an operational readiness assessment that determines when responsibility can transfer. A preliminary observation shapes the whole chapter. Regulatory data programmes fail less often because the architecture is wrong than because the implementation is unbounded. A programme that attempts to build the complete target architecture in the first delivery will not deliver the first reporting obligation on time, and the credibility lost in that failure will constrain everything that follows. The methodology presented here is therefore explicitly incremental, and its sequencing is driven by the obligation rather than by the architecture.

Implementation Phases

Figure Implementation Phases and Decision Gates presents five phases with decision gates between them. Each phase has a defined scope, deliverables and acceptance criteria, and each gate is a decision point at which progression is approved, deferred or the scope is revised.

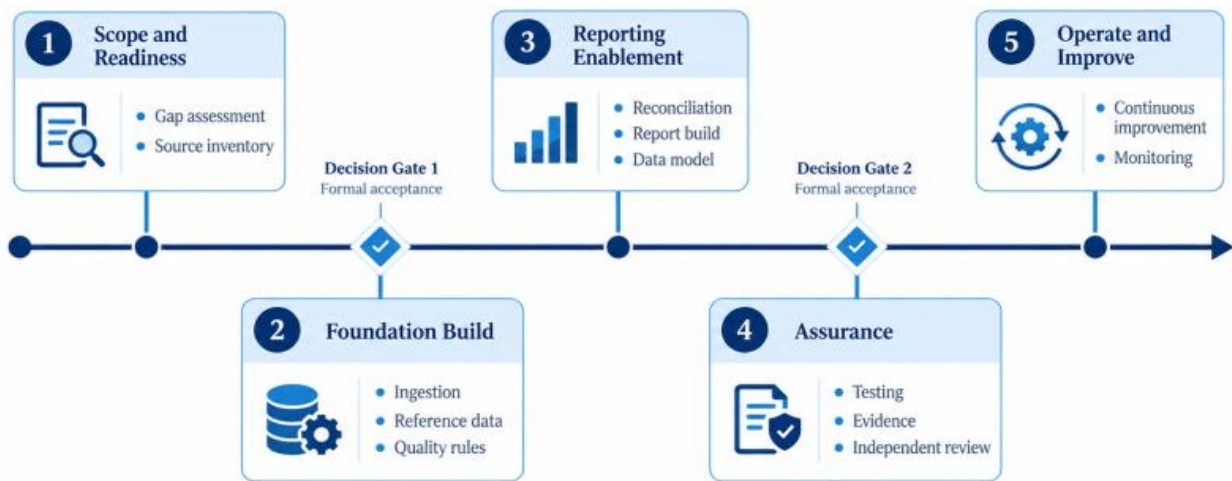


Figure 15.1 Implementation Phases and Decision Gates

Phase One: Scope and Readiness

The first phase establishes what is to be built and what the available data can support. It is the readiness assessment described in Chapter 3, executed as a project phase with defined deliverables. The deliverables are the reporting scope definition, the source inventory, the profiling results, the gap register and the readiness certificate. The acceptance criterion is that the gap register has a disposition for every gap: remediated before build, accommodated within the build, or accepted with documented limitation. The phase's principal risk is scope inflation. The inventory exercise reveals an enterprise data landscape that is more complex than anticipated, and the natural response is to expand the assessment. The discipline required is to assess what the current obligation requires and to record the wider findings for later obligations without pursuing them.

Phase Two: Foundation Build

The second phase builds the shared foundations: ingestion for the sources in scope, the reference and master data structures required, the quality rule framework, and the lineage capture. The phase is genuinely foundational in the sense that the later phases depend on it and cannot proceed around it. The deliverables are the operational ingestion pipelines with their completeness controls, the governed reference data sets in scope, the validation rule inventory with its execution framework, and the lineage coverage for the data in scope. The acceptance criterion that matters is that the foundations operate end to end on real data for a complete period before reporting logic is built on them. A programme that builds reporting logic on foundations that have not been exercised will discover the foundation defects during reporting testing, when the cost of correction is highest.

Phase Three: Reporting Enablement

The third phase builds the reporting data model, the transformation rules that populate it, the reconciliation framework and the report generation and rendering. The deliverables are the reporting data model, the rule inventory with its effective dating, the reconciliation specifications with their results, and the generated returns for at least one period, reconciled against the current production process. The reconciliation against the current production process is the phase's most valuable output and should be treated as its principal acceptance criterion rather than as a testing detail. Where the new platform's output can be reconciled to the existing output line by line, the institution has established that the new platform produces the same results and understands why any differ. Where it cannot, the institution does not know whether the new platform is correct.

Phase Four: Assurance

The fourth phase performs the testing, evidence assembly and independent review described later in this chapter. The deliverables are the test evidence by layer, the control operation evidence for a complete period, the reconciliation statements and the independent review report. The acceptance criterion is that the platform has produced a complete reporting cycle with its controls operating and its evidence assembled, and that an independent reviewer has concluded that the evidence supports the conclusion.

Phase Five: Operate and Improve

The fifth phase transfers responsibility to the operational organisation and establishes the continuous improvement mechanisms described in Chapters 10 and 14. The deliverables are the run books, the monitoring and observability configuration, the exception queues with their ownership, and the governance artefacts established in Chapter 13.

Sequencing Under Constraint

The phases are sequential in principle and overlapping in practice. The overlap that is safe and the overlap that is not should be distinguished. It is generally safe to begin reference data work before ingestion is complete, because reference data can be prepared independently of the pipelines that will consume it. It is generally unsafe to begin reporting logic before ingestion and quality frameworks are operating, because the reporting logic will be designed against foundations whose behaviour is not yet known. The practical rule is that a phase may begin when its dependencies are operating rather than when they are complete. Ingestion for the sources in scope must be operating for the reporting phase to begin, but ingestion for sources outside the current scope may be added later without disrupting what has been built, provided the canonical model has been designed to accommodate them.

Testing Strategy

Figure Assurance and Testing Layers presents the assurance layers. The layering reflects a principle: testing effort should be distributed so that each layer tests what it is capable of testing, and no layer is asked to compensate for gaps in another.

Static and Structural Validation

The base layer verifies that the artefacts are well formed and conform to their specifications: that schemas match their definitions, that rules parse, that mappings are complete against the target, that references resolve and that identifiers are unique where they must be. The layer is cheap and should be exhaustive, because it catches a class of defect that is otherwise detected only in later testing at much higher cost. A mapping that omits a target data point is trivially detectable at this layer and expensive to detect after reporting logic has been built on the assumption that the mapping was complete.

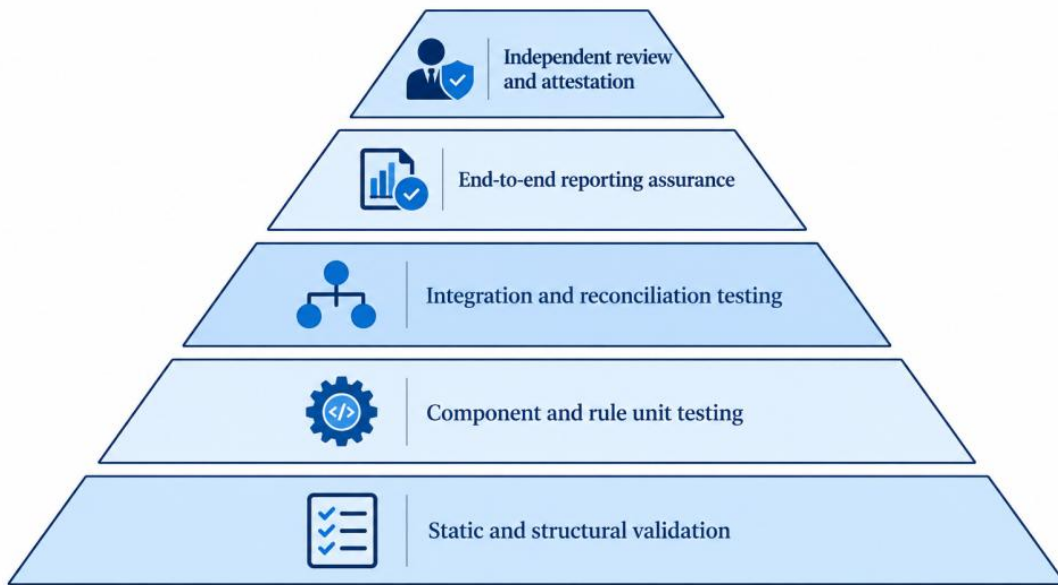


Figure 15.2 Assurance and Testing Layers

Component and Rule Unit Testing

The second layer tests individual rules against constructed inputs with known expected outputs. Its distinctive value is that it tests the rule in isolation, so that a failure identifies the rule as the cause. Three classes of input are required. Typical inputs verify that the rule behaves as expected on ordinary data. Boundary inputs verify behaviour at the edges of each condition, which is where derivation errors cluster. Defective inputs verify that the rule handles the defects it is designed to address, which is the class most frequently omitted. A cleansing rule tested only on clean data has not been tested.

Integration and Reconciliation Testing

The third layer tests the assembled pipeline end to end: that data flows from source to output, that the controls execute, and that the results reconcile. The distinctive discipline of this layer is testing with deliberately introduced defects. A validation rule that has never been shown to detect the defect it addresses cannot be relied upon. The technique is to inject a known defect at a known point, execute the pipeline and confirm that the control detects it and that the exception is routed correctly. The exercise is inexpensive and it tests the detection mechanism, the routing and the exception handling simultaneously.

End-to-End Reporting Assurance

The fourth layer tests the complete reporting cycle, including the human steps: the review, the approval, the submission and the post-submission handling. The layer is distinctive because it tests the process rather than the system, and process defects are frequently the ones that produce deadline failures. The layer should be tested against a real period rather than a synthetic one, and the testing should include a comparison against the previously reported figures for that period. Where the comparison produces differences, each must be explained, and the explanation must distinguish differences that are corrections of known prior errors from differences that indicate a defect in the new platform.

Independent Review and Attestation

The fifth layer provides assurance by a party independent of the build. Its purpose is to test the propositions that the build team is least able to test about itself: that the controls are designed appropriately, that they operate as designed, and that the evidence supports the claims made about them. The independent review should be scoped to the control framework and the evidence rather than to the code. Reviewing code establishes that the code does what it says; it does not establish that what it says is correct. The questions the review should answer are whether the controls address the risks, whether the thresholds are defensible, whether the exception process operates, whether the evidence is complete and whether the governance decisions were made.

Testing the Adaptability Claim

Chapter 2 noted that the adaptability expectations of the aggregation principles are commonly under-served. The testing response is a scenario test in which the platform is exercised under changed conditions. Three scenarios are useful. A volume scenario applies a multiple of the current data volume within the required processing window. A change scenario introduces a new source or a new reporting obligation and measures the effort required to accommodate it, which is the empirical test of change scalability. A failure scenario removes a source or a component during a reporting cycle and measures the institution's ability to complete the reporting. The third scenario is the most valuable and the most frequently omitted, because it tests the contingency provisions described in Chapter 12 rather than the platform's normal operation.

Operational Readiness

Figure Operational Readiness Assessment presents the operational readiness assessment as four stages of gate criteria.



Figure 15.3 Operational Readiness Assessment

Processes Documented

The first gate requires that every critical process be documented to the standard that allows a competent operator who has never performed it to execute it correctly. The standard is demanding and is correctly stated that way, because a run book that requires the reader to know what it means does not serve its purpose during an incident at three o'clock in the morning.

Controls Operative

The second gate requires that the controls be operating, with results recorded, for a complete reporting period before handover. A control that has been designed and configured but not operated for a full cycle has not been demonstrated to work.

Performance Proven

The third gate requires measured evidence of performance under realistic conditions, including the volume, change and failure scenarios described above. Claims of capacity and recovery capability should be supported by measurement rather than by architecture diagrams.

People Enabled

The fourth gate requires that the receiving organisation has the skills, the authority and the resources to operate the platform. The specific requirements are training completed, the support model agreed and documented, and the handover accepted by a named accountable person. The last element is the one that converts the readiness assessment from an assessment into a commitment. A handover accepted by a named person is a transfer of accountability; a handover implied by the absence of objection is not.

The Readiness Assessment as an Artefact

The assessment should produce a documented conclusion for each gate, with the evidence referenced and any gaps identified with an owner and a date. Where a gate is passed with gaps, the gaps should be explicit and accepted rather than omitted. A readiness assessment that concludes that everything is ready when gaps exist destroys its own value, and the pressure to produce such a conclusion at the end of a delivery programme is considerable.

Programme Governance and the Management of Contradiction

Three programme-level issues deserve treatment because they determine whether the methodology is followed. The first is the management of scope. Regulatory data programmes attract scope expansion because the platform's potential applications are broader than its immediate obligation. Expansion is legitimate when it serves a defined future obligation and illegitimate when it serves an aspiration. The discipline is to require that additional scope be matched by additional time or resource rather than absorbed, and to maintain a register of deferred scope so that it is not silently lost. The second is the management of parallel operation. Running a new platform alongside an existing one for validation is valuable and expensive. The cost is proportional to the duration, and the duration should be set explicitly at the outset with a defined exit criterion. Parallel operation without an exit criterion becomes permanent, consuming the resource that should be used to improve the platform. The third is the management of the residual. Every programme ends with items not completed. Where these are recorded with owners and dates they are a backlog; where they are not they become defects discovered by the next examination. The register of residuals is one of the most valuable artefacts a programme produces, and one of the most frequently discarded at handover.

Estimating Effort and Setting the Delivery Baseline

Effort estimation for regulatory data programmes is notoriously unreliable, and the unreliability has a specific cause: the work that is underestimated is consistently the same work. Four activities are habitually underestimated. Reference data mapping, because the mapping appears mechanical until the residual cases are encountered, and the residual cases are discovered late. Reconciliation design, because specifying the basis of a comparison requires the analytical understanding of two independently built systems. Test data preparation, because testing controls requires defective data that does not naturally exist and must be constructed. And evidence assembly, because producing the retained evidence for a complete reporting cycle is a task in its own right rather than a by-product. A programme that budgets for these four explicitly will produce an estimate closer to reality. A programme that estimates from the number of data points and the number of source systems will underestimate systematically, and the shortfall will be absorbed from testing.

The Parallel-Run Exit Criterion

Where a new platform runs alongside an existing process, the exit criterion should be agreed at the outset and expressed as an observation rather than as a belief. A usable criterion states that the new platform has produced a complete reporting cycle, with all controls operating and evidenced, for a stated number of consecutive periods, with all differences against the existing process explained and no unexplained difference above a stated threshold. The criterion is observable, its cost is quantifiable, and its completion is a fact rather than a judgement. A criterion expressed as confidence that the platform is ready is not a criterion, because it cannot be satisfied or failed and it therefore cannot end the parallel run.

Migration and Cut-Over

Migrating an existing reporting capability to the new platform is the phase in which the accuracy of the plan is tested, and where the accumulated simplifications of the design become visible. Two migration strategies are available. The big-bang approach builds the complete new capability and switches on a single reporting cycle. Its advantage is that no period is processed by two systems in parallel, and its disadvantage is that the first production cycle exercises every component at once, so that a failure in any of them delays the reporting obligation. The incremental approach migrates obligations one at a time, or data domains one at a time within an obligation. Its advantage is that each step is bounded and its failures are contained; its disadvantage is that the institution operates two platforms concurrently, with the reconciliation between them becoming a required control. The choice should follow the institution's tolerance for a delayed submission, which is a function of the reporting frequency: a quarterly obligation with a long submission window tolerates more migration risk than a daily one. Where the incremental approach is chosen, the reconciliation between the two platforms must be

designed, operated and evidenced from the first migrated obligation, because it is the only control that establishes that the two are producing the same figures.

Testing Controls, Not Only Code

A testing programme that validates the platform's transformations without validating its controls produces a platform that computes correctly and cannot demonstrate that it does. Control testing follows a sequence. Each control must be shown to execute, evidenced by a result record in the environment under test. It must be shown to fail when it should, by injecting data that the control is intended to detect and confirming that it reports a breach, which is the test most frequently omitted and the one that establishes the control is not vacuous. It must be shown to produce the correct exception classification and severity, because a control that detects a defect and routes it as a low-severity item is operationally equivalent to a control that does not fire. Its threshold must be demonstrated to have been set from its stated basis, by reproducing the derivation. And its evidence output must be complete, containing the attributes a reviewer would need. The second test proving that a control fails is inexpensive and it is the only test that distinguishes a control from a metric.

Operational Readiness as a Set of Demonstrated Capabilities

Operational readiness is frequently assessed as a matter of confidence. It is better assessed as a set of capabilities that have been demonstrated in the environment that will operate them. The capabilities that should be demonstrated before the first production cycle are: a full reporting cycle executed end-to-end in the production environment with production-like volumes; a documented and rehearsed recovery from a mid-cycle failure; a completed reprocessing of a historical period, timed; a complete evidence pack assembled and reviewed by the function that will maintain it; a named operator for every routine process with a defined escalation path; and a working exception process that has processed at least one synthetic exception end-to-end. Each demonstration produces an artefact, and the artefacts together constitute the readiness evidence. An assessment conducted by review of documentation produces an opinion; an assessment conducted by demonstration produces evidence, and the difference is what makes the evidence defensible in the first cycle's review [31].

What Acceptance Should Require

Acceptance is frequently granted on the basis of a demonstration of functionality, and the criteria that determine whether the platform can be operated are not functional. Six acceptance criteria describe operational capability. The platform computes every reported figure of the in-scope obligations and produces the evidence for each, demonstrated in the operating environment. Every control executes and produces a result record, and at least one control per class has been shown to fail when given data it should detect. The reporting cycle has been completed end-to-end within the period allowed, including the approval and release steps. A historical period has been reprocessed from retained data within a stated time. Every routine process has a named operator and a documented procedure that has been executed by someone other than its author. And the exception process has processed a set of synthetic exceptions from detection to closure, producing the artefact set that an examination would request. The last two criteria are the ones most often waived and the ones that determine whether the first production cycle is calm or chaotic, because they establish that the institution can operate what it has built rather than only that it has been built.

Knowledge Retention and the Bus Factor

A regulatory data platform whose behaviour is understood by one or two individuals is exposed to a risk that no architectural control addresses and that examinations occasionally reveal. Three practices reduce it. Procedures are written as executable steps rather than as descriptions, so that a procedure can be followed by someone who has never performed it. Design decisions are recorded with their rationale, not merely their outcome, so that a later maintainer can tell whether a decision still applies. And rule documentation is derived from the rules rather than maintained alongside them, so that the documentation describes the current state rather than the state at the time of writing. The measure is the time required for a competent engineer who is new to the platform to make a defined change and demonstrate that it is correct. Where the measure is long, the platform's cost of change is high regardless of its architecture, because the constraint is knowledge rather than design.

CHAPTER 16

Architectural Trade-Offs, Integrated Case Studies, and Future Directions

The preceding chapters have presented a reference architecture, the controls that sustain it and the methodology by which it is implemented. This final chapter does three things. It makes the trade-offs explicit, presenting the design space as a set of positions with consequences rather than as a single correct answer. It presents integrated case studies illustrating the architecture in operation. And it examines the directions in which requirements and capabilities are moving, with the caveat that predictions are the least reliable element of any technical work.

The Trade-Off Framework

Figure Architectural Trade-Offs in Regulatory Data Platforms positions the principal architectural choices against two axes that dominate the economics of a regulatory data platform: the time required to deliver the first reporting cycle, and the cost of change in later cycles.

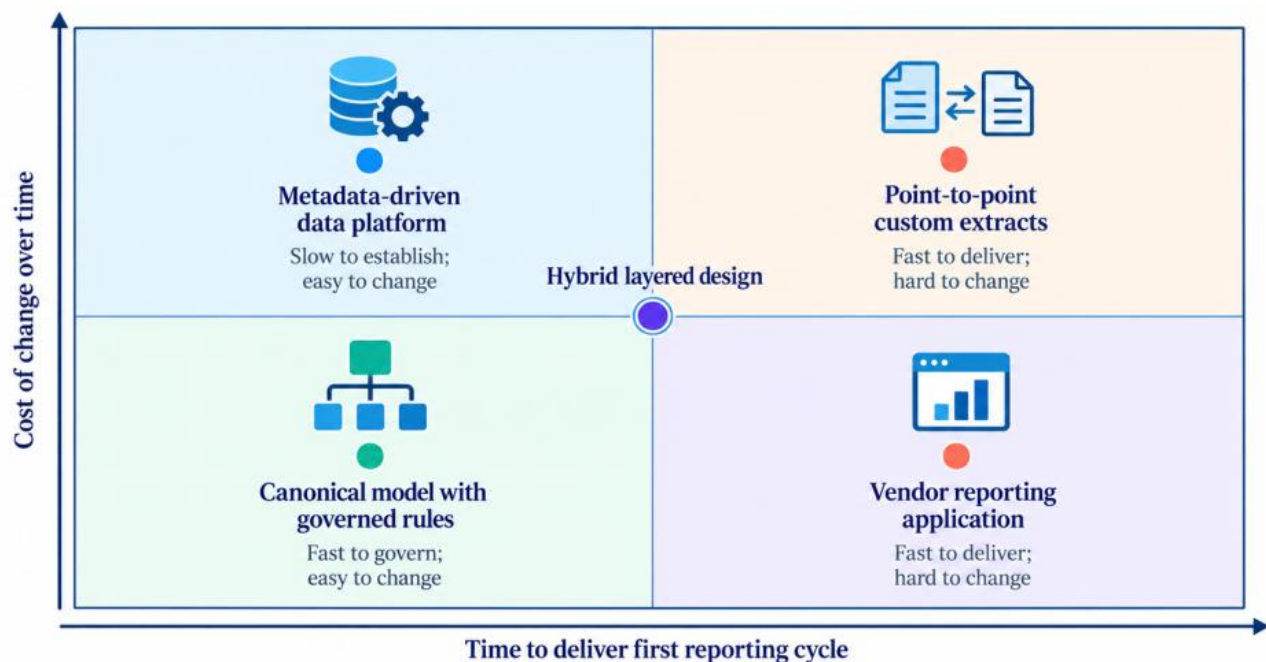


Figure 16.1 Architectural Trade-Offs in Regulatory Data Platforms

The Two Axes

The first axis recognises that regulatory data platforms are delivered under delivery pressure. The obligation has a date. A design that takes longer to deliver its first cycle imposes a cost in delay that must be justified by something else. The second axis recognises that a platform built once is modified continuously. Reference data changes, frameworks change, sources are replaced, obligations are added. The cost of the two hundredth change is a real cost and it is frequently invisible at the time the architecture is chosen. The two axes are not independent, and their interaction produces the characteristic positions shown in the figure.

Point-to-point custom extracts deliver quickly and are expensive to change. Each new obligation produces a new set of extracts and transformations, each with its own logic and its own defects. The approach is rational for a single, stable, small obligation and becomes untenable as obligations multiply.

A canonical model with governed rules takes longer to deliver and is inexpensive to change, because a new obligation reuses the conformed data and adds only its presentation rules. The approach requires an upfront investment in semantic design and governance that produces no visible benefit in the first cycle.

A vendor reporting application delivers quickly for the obligations it supports and is expensive to change outside them. Its economics depend on the fit between the vendor's coverage and the institution's obligation profile, and on whether the institution's differentiated requirements are accommodated by configuration or by customisation. Customisation converts a fast-to-deliver option into a hard-to-change one.

A metadata-driven platform occupies the middle position and its economics depend on scale. The investment in the rule repository, the execution engine and the lineage capture is substantial and is repaid only when the number and the change rate of the rules justify it.

Making the Trade-Off Explicitly

The framework's purpose is not to rank the options but to force an explicit choice. The failure mode it addresses is the implicit choice: an institution that builds point-to-point extracts because that is what the first obligation required, and discovers three obligations later that it has built a structure it cannot afford to maintain. The practical instrument is a statement, made at the outset and revisited at each major obligation, of the institution's position on the two axes and the reasons for it. An institution that consciously chooses fast delivery with high change cost, because its obligations are few and stable, has made a defensible decision. An institution that arrives at the same design without having considered the alternative has not made a decision at all.

Secondary Trade-Offs

Four further trade-offs recur and should be decided explicitly.

- Centralisation against federation. Discussed in Chapter 2. Centralisation simplifies control and concentrates change. Federation scales change and risks divergence. The determining factors are the number of sources, the number of obligations and the rate of change.
- Prevention against detection. Discussed in Chapter 4. Prevention is cheaper in operation and requires influence over systems outside the reporting function. Detection is achievable unilaterally and is more expensive to operate. The determining factor is the institution's authority over its sources.
- Real-time against batch. Discussed in Chapter 7. Latency is bought with operational complexity and with the loss of the bounded population that makes completeness provable. The determining factor is whether an obligation genuinely requires intraday data.
- Standardisation against flexibility. A platform that enforces a single representation for every concept gain consistency and loses the ability to accommodate legitimate variation. The determining factor is whether the variation is semantic, in which case it must be modelled, or merely representational, in which case it should be eliminated.

Integrated Case Studies

Two case studies illustrate the architecture applied at different scales. Both are presented as composite and illustrative designs rather than as descriptions of particular institutions, and the figures are illustrative rather than empirical.

Case Study One: Capital Reporting for a Retail Bank

The first case concerns an institution reporting regulatory capital. The obligation requires credit risk exposures, risk-weighted assets and capital ratios, at entity and consolidated level, with reconciliations to the general ledger.

- Source landscape. Four systems contribute: the core banking and lending platform, the collateral and guarantee register, the credit risk engine that computes exposures and risk weights, and the legal entity reference service.

Each has its own identifier scheme, its own treatment of dates and, in the case of the risk engine, its own notion of when an exposure arises.

- **Readiness findings.** The readiness assessment identifies three gaps. The legal entity hierarchy is maintained separately in the consolidation system and in the risk engine, with differences in the treatment of entities below the consolidation threshold; this is a semantic gap requiring a governed resolution rather than a data correction. Collateral valuations in the collateral register are dated by valuation date rather than by reporting date, which is a period semantics gap. The risk engine does not retain historical calculation runs, which is a reproducibility gap requiring the retention of inputs and rule versions rather than outputs.
- **Architecture.** The platform ingests the four sources into an immutable landing zone with control totals captured at extraction. A reference data hub holds the legal entity hierarchy with effective dating, and the reconciliation between the two hierarchies is performed as a governed control rather than resolved by precedence. A standardized layer conforms keys and units. A derived layer computes exposures and risk-weighted assets using effective-dated rules. An aggregation layer produces entity and consolidated aggregates. A reporting layer renders the return.
- **Controls.** Completeness is asserted against the general ledger trial balance for exposures and against the sum of loan-level balances for the loan population, using independently written queries. Position-to-transaction reconstruction verifies that exposures are consistent with movements. Record-to-record reconciliation between the risk engine and the standardized layer explains every difference by classification rule, since the two systems classify exposures differently by design. The variance explanations are archived per period.
- **Outcome.** The platform requires approximately twice the delivery effort of the point-to-point approach it replaced and reduces the effort of each subsequent obligation, because the capital return, the leverage return and several internal reports share the conformed data and differ only in their presentation rules.

Case Study Two: Insurance Reporting Under a Principles-Based Regime

The second case concerns an insurer reporting under a principles-based prudential regime, where the reporting obligation is defined by a large set of quantitative reporting templates with strict validation rules.

- **Distinguishing feature.** The distinguishing feature is the volume and structure of the reporting obligation itself. The templates contain thousands of data points with dimensional structure and with cross-template validation rules that assert relationships between data points in different templates. The reporting obligation effectively specifies a data model, and the institution's model must be reconcilable to it.
- **Architecture.** The platform is built around a target-state mapping from the institution's data model to the reporting taxonomy, with the mapping maintained per taxonomy release. Reference data includes the classifications the taxonomy requires, mapped with explicit treatment of residual cases. The reporting layer is generated from the taxonomy definition rather than hand-built, so that a taxonomy release produces a change in the mapping and in the generated presentation rather than a rebuild.
- **Controls.** The cross-template validation rules are executed as a distinct control layer, since they test relationships that no single-template validation can detect. A change in one data point that breaks a cross-template relationship is detected by this layer and would otherwise be invisible.
- **Readiness findings.** The readiness assessment identifies that the institution's product taxonomy does not map cleanly to the reporting taxonomy for a small number of legacy products, requiring governed decisions rather than automated mapping. This is recorded as an accepted limitation with a documented decision for each affected product, and the decisions are maintained as reference data.

What the Cases Illustrate

Three generalisable observations emerge. First, in both cases the material findings were semantic rather than technical: hierarchies maintained in two places, taxonomies that do not map cleanly, period semantics that differ between systems. Technical defects are easier to detect and cheaper to fix. Second, in both cases the control that delivered the most assurance was a reconciliation against an independent basis, not a validation rule. Validation confirms that the institution's data conforms to the institution's expectations; reconciliation detects the consequence of expectations the institution had not articulated.

Third, in both cases the investment that most reduced the cost of subsequent obligations was the conformed reference data layer. It is the layer most frequently deferred and the one whose deferral is most reliably repaid in rework.

Directions of Change

The requirements and the available capabilities are both moving. Four directions are visible from the position of 2020, and they are presented as expectations rather than as predictions.

Granularity and Frequency

Supervisory expectations have moved consistently in the direction of finer granularity and shorter cycles. Where reporting was periodic and aggregated, it is increasingly granular and continuous, whether through formal obligations or through ad hoc data requests that assume the institution can produce granular data on demand. The architectural implication is that the granularity decision described in Chapter 11 should be taken with the expectation of future demands rather than current ones, and that the retention of granular history should be treated as an asset rather than as a cost.

Traceability as a Default

Expectations about traceability have also moved in one direction. It is increasingly assumed that an institution can demonstrate, for any reported figure, its source attributes, the rules applied, the reference data versions in force and the approvals obtained. The expectation has shifted from responding to a question over weeks to answering it over hours. The architectural implication is that lineage capture, rule versioning and reference data versioning are becoming baseline capabilities rather than enhancements. Their cost is incurred once and their absence becomes progressively more expensive.

Governed Automation

Automation of data processing is mature; automation of control is less so. The direction of travel is toward controls that are configured declaratively, executed automatically and evidenced automatically, with human attention reserved for exceptions and for the design of the controls themselves. The architectural implication is that the metadata-driven approach of Chapter 8 becomes more rather than less valuable, because it is the substrate on which automated control depends. It also raises the governance question of who is accountable for a control that no person executes, which the model in Chapter 13 answers by assigning ownership to the control's designer rather than to its operator.

Convergence of Analytical and Regulatory Data

The final direction is the convergence of the data used for regulatory reporting with the data used for internal analytics. The convergence is driven by economics: the same granular, governed, traceable data serves both, and maintaining two versions is duplicative. The convergence is not complete, because the requirements differ. Regulatory reporting requires defined semantics, controlled change and exact reproducibility. Analytics requires flexibility, rapid iteration and tolerance of imprecision. The practical architecture that reconciles them is a shared conformed and derived layer with distinct consumption paths: a controlled path for reporting and a permissive path for analytics. The pattern preserves the regulatory guarantees while allowing analytical use, and it requires that the permissive path be prevented from feeding back into the controlled path without going through the governed change process.

A Concluding Assessment

The architecture described in this book is more expensive to build than the approach it replaces and less expensive to operate, to change and to defend. Whether that exchange is favourable depends on the obligation profile and the change rate, and the framework in this chapter is offered as the instrument for making that judgement explicitly. Three propositions do hold generally. Control is cheaper when it is placed close to origin, and the cost of a defect increases with the distance between its introduction and its detection. Traceability is cheaper when it is captured than when it is reconstructed, and its reconstruction becomes impossible once the code and reference data that produced a figure have changed. And the cost of change dominates the cost of construction in any platform that is used for more than a single reporting cycle, which means that the architectural choices that determine changeability deserve more attention at design time than they typically receive. The engineering task is not to eliminate defects. It is to build a system in which defects are detected before they are reported, in which correct results can be demonstrated rather than asserted, and in which the

accumulated understanding of how the numbers are produced survives the departure of the people who produced them. That is what it means to engineer regulatory data.

A Decision Procedure for the Trade-Off

The framework in this chapter is more useful when reduced to a sequence of questions that a design team can answer in a workshop. What is the **obligation profile**? How many obligations will this platform serve over the next five years, and how similar are they in their data requirements? A single stable obligation argues for a direct design; several similar obligations argue for a conformed layer serving projections. What is the **change rate**? How often do the sources, the frameworks and the obligations change? A low change rate makes construction cost dominant and favours fast delivery. A high change rate makes change cost dominant and favours the conformed, metadata-driven design. What is the institution's authority over its sources? Where the reporting function can impose requirements on source systems, prevention is available and cheap. Where it cannot, detection is required and its cost must be budgeted. Does any obligation require intraday data? If none does, streaming ingestion is an unnecessary complexity. If one does, it should be scoped to the obligation rather than applied platform-wide. What is the evidential burden? How likely is the platform to be examined, and how specific are the questions likely to be? A high evidential burden justifies lineage capture, rule versioning and evidence retention as baseline capabilities rather than enhancements. The sequence produces a defensible position rather than a unique answer, and its value is that the position is stated and can be revisited when the answers change.

Closing: The Obligation to Be Explainable

The regulatory reporting obligation is, at its core, an obligation to be able to explain. An institution must be able to explain what it reported, why it was correct when it was reported, and how it knows. The engineering described in this book exists to make those explanations available as routine outputs of a functioning system rather than as the product of an investigation. That is the standard against which the architecture in this book should be judged, and it is a standard that can be tested rather than asserted: choose a reported figure from a past period and ask how long it takes to produce a defensible account of how it was derived and why it is believed to be correct. The answer is the measure of the architecture.

An Integrated Case: Building the Aggregation Layer First

Consider an institution with four reporting obligations of different frequencies, a governance function that has decided to build a governed aggregation layer before any of them, and a supervisory deadline that requires the nearest obligation to be submitted for the next quarter. The sequencing that satisfies both proceeds in five steps. The obligation inventory is produced first, since it determines which data domains will be reused and therefore which parts of the aggregation layer must exist immediately. The minimum governed core is then defined: the data domains required by the nearest obligation and by at least one other, since a domain used by one obligation alone can reasonably be served directly while a domain used by several justifies the conformed treatment. The mapping layer is built next, because it is the artefact that makes the subsequent obligations cheaper and because it is the artefact whose absence is most expensive to remedy. The obligation-specific projection for the nearest obligation is built on top, and the remaining obligations are delivered subsequently against the established core. The design decision that makes the sequence work is that the minimum governed core is defined by reuse rather than by domain completeness. A programme that attempts to complete the aggregation layer before delivering any obligation places the deadline at risk; a programme that delivers the nearest obligation outside the core forfeits the reuse that justified the core. Defining the core by the domains that more than one obligation requires resolves the conflict, and it does so at design time, when the inventory is being produced, rather than at delivery time, when the choice is between slipping a deadline and duplicating an extract.

An Integrated Case: Retrofit of Lineage to an Existing Platform

A second case concerns an institution whose platform predates its lineage obligations and which has twelve hundred transformation scripts whose dependencies are undocumented. The retrofit proceeds by declarative capture where possible and inference where not. Scripts that already declare their inputs and outputs can be registered by reading those declarations, which is cheap. Scripts expressed in procedural form must be parsed or annotated, and the annotation approach requiring the developer to record the mapping as an attribute of the script is more reliable than parsing and is sustainable if it is enforced at commit. The critical decision is the entry point of the retrofit. Retrofit is often attempted from the source systems inward, which produces a graph whose completeness cannot be established and whose construction consumes the whole programme. Retrofitting from the reported data points backward is bounded by the

obligation, produces a graph that covers what is reported, and leaves the non-reported transformations to be captured opportunistically. Coverage is then a measurable proportion of reported data points rather than an unmeasurable proportion of the estate [27], [28].

Where the Field is Heading

Three directions are visible and each is stated here as an expectation rather than as an established finding. The first is the convergence of lineage and control. Lineage is currently maintained as a description of data movement and controls as an operational artefact, and the two use overlapping but distinct representations. Where the same declarative artefact defines both the transformation and the control that asserts the transformation's precondition and postcondition, the model of the pipeline becomes substantially more coherent, and the problem of controls that are documented in a register but not connected to the pipeline disappears [19]. The second is the rise of the data contract as the integration artefact. The interface contract described in Chapter 2 is currently a document maintained by agreement; its expression as an executable specification that the producing and consuming systems both validate against converts it into a testable artefact. The expectation is that contracts will increasingly be versioned, published and enforced at runtime rather than reviewed at design time [63]. The third is the specialisation of governance for regulatory data. Data governance has developed general frameworks for stewardship, quality and policy [34], [35]. The obligations that apply to regulatory reporting reproducibility of a past figure, defensibility of an adjustment, completeness against an independent basis impose requirements that general frameworks do not articulate, and the expectation is that these requirements become explicit elements of the governance framework rather than being inferred from it. None of the three reduces the number of principles that must be satisfied. Each changes the practical cost of satisfying them, and an architecture that is explicit about its transformations, its controls and its contracts is the architecture best positioned to benefit [49].

A Summary of Design Positions

The positions this book takes are stated here in a form that can be used as a checklist during a design review. Each is a defensible default rather than a universal law, and each should be departed from deliberately. Govern the mapping specification as an artefact separate from the pipeline code, so that a framework revision changes a specification rather than a program. Make every transformation rule addressable by identity and version, so that the rule in force for a historical period can be recovered and a past figure reproduced. Capture lineage at the point of transformation and from the reported data point backward, so that coverage is measurable against the obligation rather than against the estate. Define every control's threshold from its stated basis rather than from the recent observed difference, and record the basis with the control. Reconcile against an independent basis, and treat a reconciliation whose two sides derive from the same path as a measurement rather than a control. Represent every adjustment as data, with its basis and its author, so that the reported figure remains a function of its inputs. Retain the execution environment, including code, rules, reference versions and configuration, so that reproducibility is a property of the platform rather than of organisational memory. Define the cut-off technically and enforce it mechanically, so that the period boundary is not a matter of discipline under deadline pressure. Size capacity for reprocessing, not for the routine cycle, because the occasions on which reprocessing is required are the occasions on which the institution can least afford to be unable to perform it. Automate the path to a validated artefact and preserve a single human authority for its release, so that accountability is located where the judgement is exercised.

The Test That Remains

The architecture described in this book is not justified by its components. It is justified by whether it can answer, quickly and with evidence, questions about what the institution reported and why. Three questions make the test concrete and can be asked of any platform at any time. Choose a reported figure from a period between one and three years ago: how long does it take to state the rule that produced it, the source attributes it consumed and the reference data versions in force? Choose a control that operates on the reporting critical path: when was it last shown to fail, and what evidence records that demonstration? Choose a data point whose population the institution cannot fully source: is that limitation documented, is a compensating control operated, and is both recorded against the data point rather than known informally? An institution that answers all three from its systems has built the capability this book describes. An institution that answers them by assembling a team has built a platform with the appearance of that capability, and the difference becomes visible at the moment it is least convenient.

Concluding Observation

Regulatory data engineering is frequently presented as a technical specialism subordinate to the business of banking or insurance. The presentation is misleading. The institution's ability to demonstrate its condition to a supervisor, to a market and to itself depends on the properties described in this book: that data is complete and traceable, that rules are governed and versioned, that controls operate and are evidenced, that figures reconcile to independent bases, and that a past position can be reconstructed. None of these properties is achieved by a single decision, and none survives neglect. Each is produced by a design that anticipates the question and by an operating routine that maintains the answer. That is the discipline, and the architecture described in these chapters is its expression. Figure Integrated Case Study: Retail Bank Capital Reporting presents an integrated case study, following the framework end-to-end for a retail bank capital reporting obligation to show how the design choices described in earlier chapters combine.

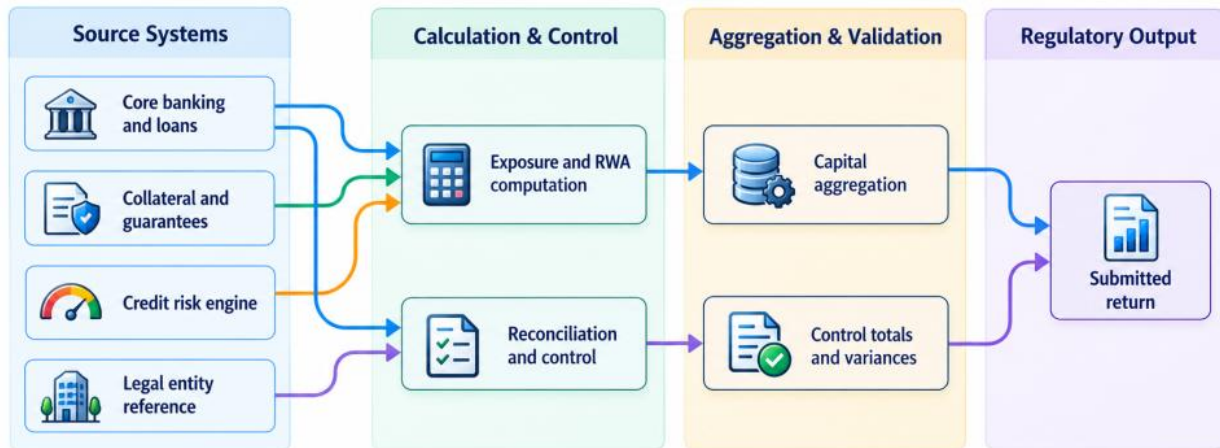


Figure 16.2 Integrated Case Study: Retail Bank Capital Reporting

Figure Directions of Change Beyond 2020 summarises the directions of change identified in this chapter, stated as expectations rather than as established findings.

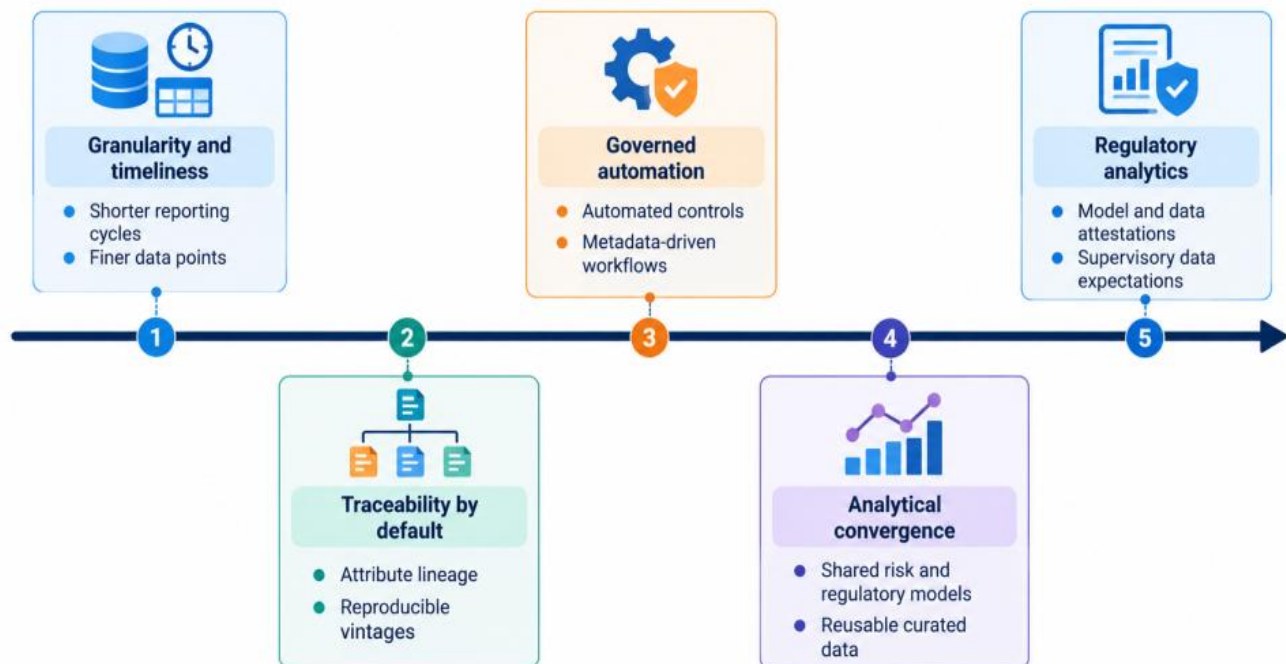


Figure 16.3 Directions of Change Beyond 2020

APPENDIX A

Glossary of Regulatory Data Engineering Terms

This glossary defines the terms as they are used in this book. Where a term has a meaning in a formal standard that differs from its general use, the standard's designation is cited. Definitions here describe the engineering sense of each term rather than the accounting, actuarial or supervisory sense, which is treated in the respective literatures.

A.1 Data and Metamodel Terms

Attribute-level lineage. The recorded relationship between a specific target column and the specific source columns that populate it, together with the transformation applied. Distinguished from table-level lineage, which records only the dependency between assets.

Binding. The recorded association between a generated artefact and the inputs that produced it: the reporting period, the data cut-off, the data version, the rule set version and the taxonomy release. Binding is what makes a produced figure explainable after the fact.

Bitemporal model. A data model that records separately the period during which a fact was true in the world and the period during which the institution believed it to be true. Required where reported figures must be reproduced as at a past date and where corrections are applied retrospectively.

Canonical model. An integration representation that is common to all sources and all consumers, introduced to avoid the combinatorial growth of point-to-point interfaces. Its cost is the design and governance of the representation itself.

Certified asset. A catalogued data asset whose definition has been agreed, whose controls are defined and operating, whose lineage is documented and for which an accountable owner has accepted responsibility.

Conformed dimension. A dimension defined once and referenced consistently across multiple reporting subjects, so that figures produced for different obligations are directly comparable.

Control total. A count, sum or hash obtained from an independent source and used as the denominator against which the completeness or accuracy of a delivered population is asserted. A control total derived from the same pipeline it tests is not independent and cannot detect systematic omission.

Critical data element. An attribute whose quality materially affects a reported figure, identified so that control investment may be prioritised to it.

Data cut-off. The point in the reporting calendar after which arriving data is not included in the current return. Its consistent application, and the quantification of what it excludes, is what makes it a specification rather than an operational variable.

Data readiness. The extent to which available data, in its existing condition and location, can support a specified obligation without unplanned remediation. Readiness is relational: the same dataset may be ready for one obligation and not for another.

Denominator problem. The structural weakness in which a completeness or accuracy measure derives its expectation from the same path it is testing, so that numerator and denominator fail together and the control reports success.

Effective dating. The association of every governed artefact — a reference code, a transformation rule, a hierarchy relationship — with the period for which it holds. The precondition for reproducing a report for a past period.

Golden record. The institution's governed authoritative representation of an entity, assembled from the records of multiple source systems by survivorship rules. It is a governed composite rather than a truth claim.

Hard stop. A control failure that halts processing because the output is not meaningfully interpretable a file that will not parse, a control total that does not agree. Distinguished from a soft flag, which permits processing with an attached quality indicator.

Idempotence. The property of a process that re-executing it produces the same state rather than duplicating its effect. Required for safe recovery in ingestion and transformation.

Immutable landing zone. The store in which data is written as received and never modified, so that corrections arrive as additional files and both the original state and the correction are retained.

Lineage. The record of the relationships through which data moves and changes, at table, column or row granularity, optionally extended with derivation semantics as provenance.

Period semantics. The definition of the period that a source delivery actually represents. Because the label on a file does not determine this, period semantics must be established empirically per source and recorded as specification.

Provenance. A record of the derivation of a value that includes not only which source contributed but through which transformation and under what conditions. The literature distinguishes why-provenance, how-provenance and where-provenance.

Reference data. The controlled sets of values used to classify and describe other data, characterised by being relatively static, standardised and shared across many consumers.

Master data. The core entities about which the enterprise transacts, characterised by being large in population and requiring identity resolution across independently built systems.

Restatement. The correction of a return already submitted, whether required by the authority or initiated by the institution. Sensitive because it is externally visible, and requiring governance of cause, effect and approval.

Rules that are data. The architectural principle in which transformation logic is expressed as governed specifications interpreted by a generic engine, so that lineage, versioning, impact assessment and testing follow from the specification rather than from reverse engineering.

Soft flag. A quality indicator attached to a record that has failed a non-critical expectation, allowing the record to proceed into the reported population with its deficiency explicitly recorded.

Survivorship. The set of attribute-level rules by which the values of a golden record are selected from the contributing source records.

Taxonomy release. A versioned publication of the structure and content of a reporting framework, in machine-readable form, against which a return is validated and rendered.

Traceability. The institutional capability of answering, for any reported figure and any past period, which source attributes contributed, which rules were applied, which reference data versions were in force and who approved the release.

Waiver. The documented and time-limited acceptance of an exception without remediation, approved by an identified authority. Distinguishable from remediation, and its accumulation is a control signal rather than an accepted state.

A.2 Regulatory and Standards Terms

BCBS 239. The Basel Committee's principles for effective risk data aggregation and risk reporting, published in 2013. Expressed as supervisory expectations rather than technical specifications, and therefore requiring institutional translation into capabilities and evidence.

COSO. The Committee of Sponsoring Organizations internal control framework, refreshed in 2013. Its distinction between entity-level, process-level and transaction-level controls maps onto layered data architectures, and its evidence requirement maps onto control operation records.

IFRS 9. The financial instruments standard introducing an expected credit loss impairment model. Its data requirements reach deeply into transaction-level history, forward-looking economic scenarios and staging logic, and it is a frequent driver of data lineage investment.

ISO 8000. The international data quality standard series. Part 61 defines a process reference model for data quality management; part 110 addresses the exchange of master data characteristics; part 2 defines vocabulary.

ISO/IEC 25012. The data quality model within the SQuaRE series, defining a general model for data retained in structured form and distinguishing inherent, system-dependent and dependent characteristics.

ISO 20022. The universal financial industry message scheme, providing a metamodel for message definitions and a growing library of specific standards used in payments and securities messaging.

Legal entity identifier. The globally unique identifier for entities engaging in financial transactions, defined by ISO 17442. Its adoption inside an institution is frequently partial, with the standard used at the interface while internal representations remain proprietary.

SDMX. The statistical data and metadata exchange framework, used for the exchange of statistical data between authorities and reporting institutions.

SR 11-7. Supervisory guidance on model risk management. Relevant to data architecture where reported figures depend on models, since its validation requirements extend to a model's data inputs.

XBRL. Extensible business reporting language, providing a syntax for structured business reporting with an accompanying taxonomy mechanism. Used widely for prudential and financial reporting submissions.

Terms Relating to Control and Assurance

Control. An automated or manual procedure that evaluates data or a process against a stated expectation and produces a recorded result. A procedure that evaluates without producing a result is a measurement rather than a control, since no evidence of its operation survives.

Control total. An independently maintained aggregate used as a basis for asserting completeness of a population. A control total published by a process downstream of the extract it is compared against does not serve as an independent basis.

Compensating control. A control that provides assurance over a risk in place of, or in addition to, the primary control, typically where the primary control is impracticable. The acceptance of a compensating control requires the risk it does not cover to be stated.

Materiality. The magnitude at which a misstatement would be expected to influence a user of the reported information. An attribute of the reported figure, determined by the framework and the institution's accounting policy rather than chosen in design.

Tolerance. The difference within which two independently derived figures are accepted as representing the same underlying quantity, derived from the documented structural differences between their bases.

Threshold. The value at which a control reports a result requiring an action. Distinct from tolerance in that it governs the control's behaviour rather than the acceptability of the difference.

Exception. A record produced by a control indicating that an expectation was not satisfied, carrying a classification, a severity derived from the materiality of the figures at risk, an owner and a state.

APPENDIX B

Reference Data and Reporting Standards Quick Reference

This appendix collects the standards, frameworks and reference data sets referenced throughout the book, together with the aspect of the architecture each principally constrains. It is intended as a working reference for architects who need to know which standard bears on which design decision, rather than as a substitute for the standards themselves.

B.1 Data Quality and Data Management Standards

The data quality standards define the vocabulary and the process model for data quality management. They do not prescribe an architecture, which is precisely why they are useful as a common language between the engineering function, the control function and the business.

ISO/IEC 25012 defines a general data quality model for data held in structured form, and is the origin of the separation between inherent, system-dependent and dependent data quality characteristics used in Chapter 4. It constrains the design of the validation framework: a framework whose rules cannot be mapped to the model's characteristics is likely to be addressing symptoms rather than dimensions.

ISO/IEC 25024 provides measures for the characteristics defined in ISO/IEC 25012. Its relevance is to threshold design: it offers standardised measurement definitions that prevent an institution from inventing a completeness measure that is not comparable over time or across domains.

ISO 8000-61 defines a process reference model for data quality management. Its practical value is that it separates the activities an organisation must perform x data quality planning, control, assurance and improvement — from the technology used to perform them. It constrains the operating model rather than the platform.

ISO 8000-110 addresses the exchange of master data characteristics, specifying syntax, semantic encoding and conformance requirements. It constrains the interface between the master data layer and its consumers.

ISO 8000-2 defines the vocabulary of the series, and is the reference to cite when a definition of a quality term is contested.

The DAMA-DMBOK is a body of knowledge rather than a standard, but functions as a de facto reference framework for the scope of data management. Its value in this context is its decomposition of data management into knowledge areas, which provides a conventional way of locating a given engineering activity within the wider discipline.

B.2 Reference Data and Identification Standards

Table B.1 Reference data and identification standards with their architectural constraints

Standards	Subject	Architectural constraint imposed
ISO 17442	Legal entity identifier	Requires a mapping from internal entity identifiers to LEIs, governed as reference data with completeness control against the published code population
ISO 4217	Currency codes	Requires currency to be stored as a coded value rather than free text, with a mapping to internal currency identifiers where they exist
ISO 3166	Country codes	Requires the distinction between country of risk, booking, residence and incorporation to be modelled explicitly rather than collapsed into a single attribute

Standards	Subject	Architectural constraint imposed
ISO 8601	Date and time representation	Requires dates and intervals to be represented unambiguously, including time zone and duration semantics
ISO 20022	Financial message metamodel	Constrains the definition of message-based interfaces and requires taxonomy versioning to be accommodated by the release process
XBRL	Structured business reporting	Requires the presentation layer to be generated from a taxonomy rather than hand-built, so that a release produces a mapping change rather than a rebuild
SDMX	Statistical data and metadata exchange	Requires data and its structural metadata to be exchanged together, which reinforces the separation of data from presentation semantics
ISO 3166 subdivisions	Sub-national codes	Relevant where obligations require reporting below country level, and frequently omitted from mapping tables until first needed

B.3 Risk, Control and Supervisory Frameworks

BCBS 239 is the most directly architectural of the supervisory publications. Its fourteen principles address governance, data architecture, accuracy and integrity, completeness, timeliness and adaptability. Table 2.1 in Chapter 2 translates the principal areas into architectural capabilities and evidence artefacts, and that translation is the organising spine of Chapters 2, 6, 9 and 14.

SR 11-7 addresses model risk management. Where a reported figure depends on a model, the guidance requires that the model be validated, that its inputs be validated and that the model be subject to change control. The data architectural consequence is that model inputs must be versioned and traceable, and that model outputs must be retained with the input versions that produced them.

The COSO internal control framework defines the general approach to internal control over reporting. Its relevance is that it establishes the expectation that controls be documented, operated and evidenced, which is the requirement that motivates the control operation records described in Chapters 4 and 12.

Solvency II imposes a principles-based prudential reporting regime for insurers, characterised by a large set of quantitative reporting templates with cross-template validation rules. Its structural feature an obligation that effectively specifies a data model is the subject of the second case study in Chapter 16.

IFRS 9 introduces an expected credit loss impairment model whose data requirements reach into transaction-level history and forward-looking scenarios. It is a frequent driver of lineage investment because its outputs must be explainable to auditors in terms of the inputs that produced them.

The General Data Protection Regulation constrains the processing of personal data through the principles of lawfulness, purpose limitation, data minimisation, accuracy, storage limitation, integrity and accountability. Chapter 13 addresses its reconciliation with reporting obligations, which arises principally in relation to the granularity of retention and the treatment of historical reporting data that must reflect the state as at the reporting date.

B.4 Security and Privacy Frameworks

ISO/IEC 27001 defines requirements for an information security management system, providing the governance frame within which the technical controls of Chapter 13 are operated.

NIST SP 800-53 provides a comprehensive catalogue of security and privacy controls, of which the access control, audit logging and personnel security families bear most directly on a regulatory data platform.

The NIST cybersecurity framework organises security activity into identify, protect, detect, respond and recover functions, and its detect and respond functions map closely onto the observability signal inventory in Chapter 14.

Differential privacy provides a formal privacy guarantee by bounding the influence of any single individual on a published result. Its application to reported regulatory figures is constrained by the requirement that those figures be exactly reproducible and reconcilable to source records, which is why Chapter 13 positions it as relevant to analytical disclosure use cases rather than to the reported figures themselves.

B.5 Using This Reference

Three practical observations close this appendix.

First, conformance to a standard is valuable when it reduces the cost of integration, increases the credibility of evidence or supplies a vocabulary that reduces misunderstanding. It is not valuable as an end in itself, and pursuing conformance beyond what the obligation requires is a common source of wasted engineering effort.

Second, the standards that constrain architecture most tightly are the reporting and messaging standards, because they impose structure on the presentation layer and because their versioning behaviour must be accommodated by the release process. The identification standards constrain architecture more loosely, but their incomplete adoption is a frequent and avoidable source of reporting defect.

Third, the supervisory publications are the standards most frequently under-read by engineering teams, on the grounds that they are not technical. That judgement is mistaken. They define what the institution must be able to demonstrate, and a platform designed without reference to them will satisfy the technical requirements and fail the evidential ones.

Regulatory Frameworks Referenced in This Book

BCBS 239, Principles for effective risk data aggregation and risk reporting. Issued by the Basel Committee on Banking Supervision in 2013 and applicable to internationally active banks. Its principles address governance and infrastructure, data architecture and aggregation capability, accuracy and integrity, completeness, timeliness, and adaptability under stress. The framework is referenced throughout as the origin of the attribute-level lineage, independent aggregation and evidential requirements that shape the architecture described in this book [49].

Basel III. The Basel Committee's capital, leverage and liquidity framework, revised in 2017 and applicable from 2023, which determines the content of prudential returns and therefore the data domains the reporting platform must serve [50].

Solvency II, Directive 2009/138/EC. The European prudential framework for insurers, whose Pillar 3 reporting obligations impose data quality requirements on the reported populations. The European Insurance and Occupational Pensions Authority's report on data quality in Solvency II reporting is the principal analysis of the practical difficulties encountered [56], [57].

IFRS 9. The accounting standard governing the classification, measurement and impairment of financial instruments, whose expected credit loss disclosures require historical data series and behavioural assumptions that constitute a distinct data domain [58].

Regulation (EU) 2016/679, the General Data Protection Regulation. The European data protection framework, whose requirements on purpose limitation, retention and the rights of data subjects bear on the retention design of a regulatory data platform and on the boundary between the reporting population and personally identifiable data [55].

ISO/IEC 27001. The information security management standard whose control framework informs the access control and evidence architecture described in Chapter 13 [45].

ISO 31000. The risk management standard whose treatment of risk assessment and treatment informs the control design in Chapters 4 and 9 [46].

NIST SP 800-53. The security and privacy control catalogue referenced in the access control discussion, and the NIST Cybersecurity Framework referenced in the resilience discussion [62], [66].

ISO/IEC 25012 and ISO 8000. The data quality models referenced as the source of the dimension vocabulary used in Chapter 4, including the twelve data quality characteristics of ISO/IEC 25012 and the process reference model of ISO 8000-61 [39], [40], [41], [42].

XBRL and SDMX. The technical specifications for the structured representation of regulatory and statistical reporting data, referenced in the discussion of reporting data models and validation in Chapters 11 and 12 [47], [48].

ISO 20022. The financial industry messaging standard whose adoption as the basis for payment and reporting messages simplifies the translation burden in ingestion and reduces the extent of source-specific parsing required [44].

ISO 17442. The legal entity identifier standard, referenced as the basis for counterparty identification and as the principal external identifier used in entity resolution [43].

ISO 8601, ISO 3166 and ISO 4217. The date, country and currency code standards, referenced as the basis for the representation conventions applied in standardization and reference data management [67], [68], [69].

Bibliography

- [1] Y. Wang and D. M. Strong, Beyond accuracy: What data quality means to data consumers, *Journal of Management Information Systems*, vol. 12, no. 4, pp. 5–33, 1996, doi: 10.1080/07421222.1996.11518099.
- [2] D. M. Strong, Y. W. Lee, and R. Y. Wang, Data quality in context, *Communications of the ACM*, vol. 40, no. 5, pp. 103–110, 1997, doi: 10.1145/253769.253804.
- [3] L. L. Pipino, Y. W. Lee, and R. Y. Wang, Data quality assessment, *Communications of the ACM*, vol. 45, no. 4, pp. 211–218, 2002, doi: 10.1145/505248.506010.
- [4] Y. W. Lee, D. M. Strong, B. K. Kahn, and R. Y. Wang, AIMQ: A methodology for information quality assessment, *Information & Management*, vol. 40, no. 2, pp. 133–146, 2002, doi: 10.1016/S0378-7206(01)00108-3.
- [5] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, Methodologies for data quality assessment and improvement, *ACM Computing Surveys*, vol. 41, no. 3, art. 16, 2009, doi: 10.1145/1541880.1541883.
- [6] E. Rahm and H. H. Do, Data cleaning: Problems and current approaches, *IEEE Data Engineering Bulletin*, vol. 23, no. 4, pp. 3–13, 2000.
- [7] Z. Abedjan, L. Golab, and F. Naumann, Profiling relational data: A survey, *The VLDB Journal*, vol. 24, no. 4, pp. 557–581, 2015, doi: 10.1007/s00778-015-0389-y.
- [8] D. P. Ballou and H. L. Pazer, Modeling data and process quality in multi-input, multi-output information systems, *Management Science*, vol. 31, no. 2, pp. 150–162, 1985.
- [9] W. Fan and F. Geerts, Foundations of data quality, *Foundations and Trends in Databases*, vol. 5, no. 1–2, pp. 1–158, 2012, doi: 10.1561/19000000016.
- [10] L. P. English, *Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits*. New York, NY, USA: Wiley, 1999.
- [11] D. Loshin, *The Practitioner's Guide to Data Quality Improvement*. Burlington, MA, USA: Morgan Kaufmann, 2010.
- [12] K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, Duplicate record detection: A survey, *IEEE Trans. Knowledge and Data Engineering*, vol. 19, no. 1, pp. 1–16, 2007, doi: 10.1109/TKDE.2007.250581.
- [13] P. Christen, *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Berlin, Germany: Springer, 2012, doi: 10.1007/978-3-642-31164-2.
- [14] P. Fellegi and A. B. Sunter, A theory for record linkage, *Journal of the American Statistical Association*, vol. 64, no. 328, pp. 1183–1210, 1969, doi: 10.1080/01621459.1969.10501049.
- [15] M. A. Hernandez and S. J. Stolfo, The merge/purge problem for large databases, in *Proc. ACM SIGMOD Int. Conf. Management of Data*, 1995, pp. 127–138, doi: 10.1145/223784.223807.
- [16] E. Rahm and P. A. Bernstein, A survey of approaches to automatic schema matching, *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001, doi: 10.1007/s007780100057.
- [17] Doan, A. Halevy, and Z. Ives, *Principles of Data Integration*. Waltham, MA, USA: Morgan Kaufmann, 2012.
- [18] M. Lenzerini, Data integration: A theoretical perspective, in *Proc. ACM Symp. Principles of Database Systems (PODS)*, 2002, pp. 233–246, doi: 10.1145/543613.543644.
- [19] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, Conceptual modeling for ETL processes, in *Proc. ACM Int. Workshop on Data Warehousing and OLAP (DOLAP)*, 2002, pp. 14–21, doi: 10.1145/583890.583893.
- [20] W. H. Inmon, *Building the Data Warehouse*, 4th ed. Indianapolis, IN, USA: Wiley, 2005.

- [21] R. Kimball and M. Ross, *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd ed. Indianapolis, IN, USA: Wiley, 2013.
- [22] M. Kleppmann, *Designing Data-Intensive Applications*. Sebastopol, CA, USA: O'Reilly Media, 2017.
- [23] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Shelter Island, NY, USA: Manning, 2015.
- [24] M. Zaharia et al., Apache Spark: A unified engine for big data processing, *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016, doi: 10.1145/2934664.
- [25] E. A. Brewer, CAP twelve years later: How the 'rules' have changed, *IEEE Computer*, vol. 45, no. 2, pp. 23–29, 2012, doi: 10.1109/MC.2012.37.
- [26] W. Vogels, Eventually consistent, *Communications of the ACM*, vol. 52, no. 1, pp. 40–44, 2009, doi: 10.1145/1435417.1435432.
- [27] P. Buneman, S. Khanna, and W. C. Tan, Why and where: A characterization of data provenance, in *Proc. Int. Conf. Database Theory (ICDT)*, 2001, pp. 316–330, doi: 10.1007/3-540-44503-X_20.
- [28] Cheney, L. Chiticariu, and W. C. Tan, Provenance in databases: Why, how, and where, *Foundations and Trends in Databases*, vol. 1, no. 4, pp. 379–474, 2009, doi: 10.1561/19000000006.
- [29] M. D. Wilkinson et al., The FAIR guiding principles for scientific data management and stewardship, *Scientific Data*, vol. 3, art. 160018, 2016, doi: 10.1038/sdata.2016.18.
- [30] A. Zachman, A framework for information systems architecture, *IBM Systems Journal*, vol. 26, no. 3, pp. 276–292, 1987, doi: 10.1147/sj.263.0276.
- [31] Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2012.
- [32] The Open Group, *TOGAF Version 9.2: The Open Group Architecture Framework*. Reading, U.K.: The Open Group, 2018.
- [33] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. Sebastopol, CA, USA: O'Reilly Media, 2015.
- [34] DAMA International, *DAMA-DMBOK: Data Management Body of Knowledge*, 2nd ed. Basking Ridge, NJ, USA: Technics Publications, 2017.
- [35] V. Khatri and C. V. Brown, Designing data governance, *Communications of the ACM*, vol. 53, no. 1, pp. 148–152, 2010, doi: 10.1145/1629175.1629210.
- [36] J. Ladley, *Data Governance: How to Design, Deploy, and Sustain an Effective Data Governance Program*. Waltham, MA, USA: Morgan Kaufmann, 2012.
- [37] T. C. Redman, *Getting in Front on Data: Who Does What*. Basking Ridge, NJ, USA: Technics Publications, 2016.
- [38] D. Sculley et al., Hidden technical debt in machine learning systems, in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 2503–2511.
- [39] ISO/IEC 25012:2008, *Software Engineering — Software Product Quality Requirements and Evaluation (SQuaRE) — Data Quality Model*. Geneva, Switzerland: ISO, 2008.
- [40] ISO 8000-61:2016, *Data Quality — Part 61: Data Quality Management: Process Reference Model*. Geneva, Switzerland: ISO, 2016.

- [41] ISO 8000-110:2009, Data Quality Part 110: Master Data: Exchange of Characteristic Data. Geneva, Switzerland: ISO, 2009.
- [42] ISO 8000-2:2018, Data Quality Part 2: Vocabulary. Geneva, Switzerland: ISO, 2018.
- [43] ISO 17442-1:2020, Financial Services Legal Entity Identifier (LEI) Part 1: Assignment. Geneva, Switzerland: ISO, 2020.
- [44] ISO 20022-1:2013, Financial Services Universal Financial Industry Message Scheme Part 1: Metamodel. Geneva, Switzerland: ISO, 2013.
- [45] ISO/IEC 27001:2013, Information Technology Security Techniques Information Security Management Systems Requirements. Geneva, Switzerland: ISO, 2013.
- [46] ISO 31000:2018, Risk Management Guidelines. Geneva, Switzerland: ISO, 2018.
- [47] XBRL International, Extensible Business Reporting Language (XBRL) 2.1 Specification with Errata Corrections. Newark, DE, USA: XBRL International, 2003–2013.
- [48] SDMX, Statistical Data and Metadata eXchange (SDMX) 2.1 Technical Specification. Basel, Switzerland: Bank for International Settlements, 2011.
- [49] Basel Committee on Banking Supervision, Principles for effective risk data aggregation and risk reporting, Bank for International Settlements, Basel, Switzerland, Jan. 2013.
- [50] Basel Committee on Banking Supervision, Basel III: A global regulatory framework for more resilient banks and banking systems, Bank for International Settlements, Basel, Switzerland, rev. Jun. 2011.
- [51] Basel Committee on Banking Supervision, Progress in adopting the principles for effective risk data aggregation and risk reporting, Bank for International Settlements, Basel, Switzerland, 2020.
- [52] Financial Stability Board, Thematic review on risk data aggregation and reporting, FSB, Basel, Switzerland, 2014.
- [53] Board of Governors of the Federal Reserve System and Office of the Comptroller of the Currency, SR 11-7: Guidance on model risk management, Federal Reserve Board, Washington, DC, USA, 2011.
- [54] Committee of Sponsoring Organizations of the Treadway Commission, Internal Control Integrated Framework. Durham, NC, USA: COSO, 2013.
- [55] European Parliament and Council, Regulation (EU) 2016/679 (General Data Protection Regulation), Official Journal of the European Union, L 119, pp. 1–88, 2016.
- [56] European Parliament and Council, Directive 2009/138/EC (Solvency II), Official Journal of the European Union, L 335, pp. 1–155, 2009.
- [57] European Insurance and Occupational Pensions Authority, Report on data quality in Solvency II reporting, EIOPA, Frankfurt, Germany, 2020.
- [58] IFRS Foundation, IFRS 9 Financial Instruments. London, U.K.: IFRS Foundation, 2014.
- [59] International Monetary Fund, Data quality assessment framework (DQAF), IMF, Washington, DC, USA, 2003.
- [60] Cavoukian, Privacy by design: The 7 foundational principles, Information and Privacy Commissioner of Ontario, Toronto, ON, Canada, 2009.
- [61] Dwork, Differential privacy, in Automata, Languages and Programming (ICALP), 2006, pp. 1–12, doi: 10.1007/11787006_1.
- [62] National Institute of Standards and Technology, Security and privacy controls for information systems and organizations, NIST SP 800-53 Rev. 5, Gaithersburg, MD, USA, 2020.

- [63] Z. Dehghani, How to move beyond a monolithic data lake to a distributed data mesh, martinowler.com, 2019. [Online]. Available: <https://martinowler.com/articles/data-mesh-principles.html>
- [64] Z. Dehghani, Data Mesh: Delivering Data-Driven Value at Scale. Sebastopol, CA, USA: O'Reilly Media, 2022. [Post-2020 addition; not presented as contemporaneous with the 2020 reference year.]
- [65] J. Kreps, Questioning the Lambda architecture, O'Reilly Radar, 2014. [Online]. Available: <https://www.oreilly.com/radar/questioning-the-lambda-architecture/>
- [66] National Institute of Standards and Technology, Framework for improving critical infrastructure cybersecurity, version 1.1, NIST, Gaithersburg, MD, USA, 2018.
- [67] ISO 8601-1:2019, Date and Time Representations for Information Interchange Part 1: Basic Rules. Geneva, Switzerland: ISO, 2019.
- [68] ISO 3166-1:2020, Codes for the Representation of Names of Countries and Their Subdivisions Part 1: Country Codes. Geneva, Switzerland: ISO, 2020.
- [69] ISO 4217:2015, Codes for the Representation of Currencies. Geneva, Switzerland: ISO, 2015.
- [70] ISO/IEC 25024:2015, Systems and Software Engineering Systems and Software Quality Requirements and Evaluation (SQuaRE) Measurement of Data Quality. Geneva, Switzerland: ISO, 2015.

Advanced Data Engineering for Risk and Regulatory Information Systems: A Comprehensive Study of Data Readiness, Quality Controls, Reference Data, Lineage, Transformation, Reconciliation, and Regulatory Reporting Architectures provides an in-depth exploration of the data engineering practices essential for reliable risk management and regulatory reporting. The book examines key areas such as data readiness, data quality controls, reference data validation, end-to-end data lineage, data transformation, reconciliation processes, and regulatory reporting architectures. It highlights the importance of data accuracy, consistency, traceability, and governance across complex enterprise data environments, including traditional relational databases and modern Big Data platforms. By addressing data integration, validation frameworks, metadata management, and the development of dependable data pipelines, the book offers valuable insights for data engineers, data analysts, data architects, financial risk professionals, regulatory reporting specialists, and researchers seeking to strengthen enterprise data quality, improve operational efficiency, and build transparent, scalable, and reliable information systems. Designed for data engineers, data architects, financial risk professionals, regulatory reporting specialists, business analysts, technology consultants, and researchers, this book bridges essential technical concepts with the practical demands of enterprise information management. By bringing together the interconnected dimensions of data quality, governance, integration, and reporting architecture, it provides a valuable perspective on building resilient, auditable, and scalable data environments. Ultimately, this work underscores a fundamental principle of modern financial technology: reliable regulatory reporting begins with reliable data engineering.

Sudha Donepudi, Senior Data Analyst with 20 years of IT experience specializing in end-to-end data validation, requirements gathering, planning, and execution across the Financial (BFS), Telecommunications, and Healthcare industries. Proven track record of managing massive data volumes by bridging legacy data warehouses (Oracle, Teradata, DB2, Sybase) with modern Big Data environments (Hadoop, Hive, Impala). Recognized expert in FED regulatory reporting frameworks, retail risk data profiling, and account-level data quality governance. Proficient at translating complex Business Requirement Documents (BRDs) into highly optimized ETL mapping logic, maintaining data lineage, and establishing automated validation strategies using tools like QTP and QlikView.



With over 10 years of experience in Banking and Financial Services (BFS), Sudha Donepudi specializes in data analysis, data validation, and financial data architecture across hybrid environments, including Hadoop, Oracle, and Teradata. Her expertise includes gathering business requirements through Business Requirement Documents (BRDs), implementing data readiness processes, ensuring data quality, validating Federal regulatory reporting data, and maintaining end-to-end data lineage. She has extensive experience in ETL processes using Informatica and Ab Initio, SQL query optimization, UNIX and Shell scripting, and database management across Oracle, DB2, Sybase, and Teradata. Her technical proficiency also encompasses distributed file systems, Hadoop cluster data transfer and replication, high-volume data processing, and data integration to establish a Single Source of Truth (SSOT). Additionally, she is skilled in QlikView reporting, QTP automation, Hive and Impala queries, JIRA-based issue tracking, performance optimization, and enforcing enterprise architecture standards while providing technical guidance to junior engineers and stakeholders.

