FRAMEWORK · COMPANION · V1.0

The substrate, in detail

Data model, governance, authority, and the path to get there

Bearing south · Second piece · PDCA+ v2.0 · Public Review

What the substrate actually contains as a data model; how its content is governed, versioned, and changed; who holds authority over what; and the staged path an organisation takes to get from where it is today to a substrate that supports the framework's structural commitments. The piece shows the path. It does not solve the implementation.

AUTHOR

Joacim Brandell

Written for technical architects, GRC technology leads, enterprise architects, and technically-oriented CISOs · builds on Infrastructure Foundations · tool-agnostic, with entity specifications detailed enough to begin implementation

Why this piece, and the frame

The previous piece in the technical leg named what infrastructure PDCA+ requires — the database, workflow engine, event bus, policy engine, data catalog, GRC tool, artefact storage — and how they fit as a reference architecture. With the infrastructure named, the question that follows is what the substrate actually contains, how its content is governed and evolves over time, who has authority over what, and what staged path an organisation takes to get from where it is today to a substrate that supports the framework's structural commitments.

These four concerns — the data model, the lifecycle and governance of substrate content, the identity-authority-accountability structure, and the migration path — are not separable. The data model includes the entities that hold authority. The governance model determines who can change what entities. The authority structure determines whose decisions the governance respects. And the migration path is the sequenced acquisition of these capabilities in an order that allows each stage to be operationally adequate before the next is begun. Treating them as separate pieces would obscure their interdependence.

The frame for this piece, set deliberately: it shows the path; it does not solve the implementation. The data-model specifications are detailed enough that a technical architect can begin designing a real schema from them, but they do not commit to a database technology, schema syntax, or implementation language. The governance and authority structure is concrete enough to be implementable, but does not prescribe how identity systems integrate or which RBAC tool is used. The migration path identifies stages with the structural commitments each stage requires, but does not specify which months in which fiscal year. Implementation choices belong to the organisations adopting the framework, not to the framework itself.

A reader who finishes this piece should be able to begin scoping a substrate implementation — the entities to model, the relationships to capture, the authority structure to formalise, the first stage of the journey to commit to. They should also know precisely what this piece is not telling them: the products to buy, the syntax to use, the resourcing model to adopt. Those decisions are downstream of architectural commitment and are not what this piece is for.

The substrate as a data model

The substrate is a coherent collection of entities and relationships. Some entities are operational artefacts the disciplines produce and consume. Some are reference data the conjunction holds for the disciplines to address. Some are authority records that determine who can change what. Some are audit records that capture what changed when. The entities are distinguishable by what kind of data they hold and by who has authority to change them — and the distinction matters for how each is modelled, stored, and governed.

This section walks through the principal entities, specifies each one in enough detail that an architect can begin building, and names the cardinality decisions that the data model has to

make. The entities have been introduced in earlier pieces; here they receive their data-model treatment.

Operational entities versus reference entities

Before the entities themselves, a distinction worth making explicit. Operational entities are produced by disciplines as their work proceeds — risk register entries, incident records, configuration items, evidence artefacts. They have many instances; their instances are created, updated, and sometimes retired through normal operations; their lifecycle is transactional. Reference entities are conjunction-held metadata — schemas, taxonomies, translation rules, control specifications, measurement constructs. They have few instances relative to operational data; their instances change slowly through deliberate governance; their lifecycle is editorial.

The distinction matters because operational and reference entities have different storage requirements, different access patterns, different change governance, and different versioning needs. A well-designed substrate keeps them clearly separated in the data model even when they share infrastructure, and the separation is what makes the governance section that follows tractable.

The chain entities

The six chain entities from the regime-leg appendix — Regime Intent, Requirement, Control Objective, Obligation, Evidence Link, and Stakeholder Entitlement — are mostly reference entities, with one operational exception. They are specified here as a connected set, because their relationships are what makes the chain work.

ENTITY · **Regime Intent Record**

Attribute / Relation	Type / Target	Notes
intent_id	`uuid`	Primary identifier; immutable for the life of the regime
regime_name	`string`	Authoritative name (e.g. CMMC, DEFSTAN 05-138)
regime_version	`string`	Specific version this intent reflects
intent_text	`text`	What the regime is trying to achieve, in the regime's framing
jurisdiction	`string`	Regulatory jurisdiction the regime operates under
effective_date	`date`	When the regime intent took effect
retirement_date	`date · nullable`	Set when regime is retired; never deleted
requirements →	`0..n Requirement Records`	Forward relationship to requirements that decompose this intent
governance_authority	`Authority reference`	Who owns changes to this record (typically regime-side)

ENTITY · Requirement Record

Attribute / Relation	Type / Target	Notes
requirement_id	`uuid`	Primary identifier
intent_id	`Regime Intent reference`	Upstream link; required, immutable after creation
requirement_text	`text`	What the regime asks of obligated parties
regime_reference	`string`	Citation in regime's own vocabulary (e.g. CMMC RA.L2-3.11.1)
scope_conditions	`structured`	When this requirement applies (system class, data type, etc.)
control_objectives →	`1..n Control Objective Records`	Downstream link; the objectives this requirement asks for
governance_authority	`Authority reference`	Regime-side authority over this record

ENTITY · Control Objective Record · the fulcrum

Attribute / Relation	Type / Target	Notes
objective_id	`uuid`	Primary identifier
objective_text	`text`	Regime-portable articulation of what must be achieved
satisfies_requirements →	`1..n Requirement Records`	Upstream; many regimes' requirements can converge here
addressed_by_obligations →	`1..n Obligation Records`	Downstream; what the organisation commits to do
risk_treatment ←	`Risk Treatment reference`	Lateral attachment of risk-treatment decisions
measurement_constructs →	`0..n Measurement Constructs`	What adequacy looks like for this objective
governance_authority	`Authority reference`	Organisation-side authority — the objective is internal
version	`integer`	Monotonic; the objective evolves with the organisation

ENTITY · Obligation Record

Attribute / Relation	Type / Target	Notes
obligation_id	`uuid`	Primary identifier
objective_id	`Control Objective reference`	Upstream link
obligation_text	`text`	What the organisation has committed to do
responsible_role	`Role reference`	Who carries this obligation operationally
operational_context	`structured`	Tooling, scope, resourcing assumptions
evidence_links →	`1..n Evidence Link Records`	Downstream; what underwrites this obligation
governance_authority	`Authority reference`	Operational authority for this obligation

ENTITY · Evidence Link Record · operational

Attribute / Relation	Type / Target	Notes
link_id	`uuid`	Primary identifier
obligation_id	`Obligation reference`	Upstream link
artefact_reference	`URI`	Pointer to the actual evidence artefact in storage
artefact_type	`enum`	Document, log extract, attestation, scan result, etc.
effective_period	`interval`	When this evidence is valid for assurance
regime_addressability	`structured`	Which regimes' evidence-form expectations this link satisfies
created_by	`Identity reference`	Who produced this evidence link
created_at	`timestamp`	When

The Stakeholder Entitlement Record is structurally different from the others — it is a derived assertion that the substrate produces, rather than a record practitioners directly maintain. Querying the chain from any Regime Intent Record downward through current Evidence Link Records yields the entitlement; the substrate computes it rather than storing it. For audit purposes the computation may be materialised periodically, but it is fundamentally a query result.

> THE CHAIN ENTITIES, IN ONE SENTENCE
>
> *Five persistent record types — Regime Intent, Requirement, Control Objective, Obligation, Evidence Link — connected by directed relationships, with the Control Objective Record as the fulcrum where regime-side meets organisation-side; the Stakeholder Entitlement is a derived assertion computed from the chain, not a stored record.*

The supporting entities

Beyond the chain, several supporting entities are required for the substrate to do its work. Each has its own cardinality, lifecycle, and governance characteristics.

ENTITY · **Discipline Binding Record**

Attribute / Relation	Type / Target	Notes
binding_id	`uuid`	Primary identifier
discipline_name	`string`	The bound discipline (e.g. configuration management)
binding_pattern	`enum`	Full, inherited, partial, multi-strand-operator-steps
strands_contributed	`set of enum`	Which strands the discipline contributes to
services_consumed	`structured`	Which conjunction services it relies on
dimensional_axis	`structured`	Decomposition the discipline operates on
input_schemas →	`0..n Schemas`	What the discipline consumes
output_schemas →	`0..n Schemas`	What the discipline produces
effective_period	`interval`	When this binding is active

ENTITY · **Control Record · the shared inventory**

Attribute / Relation	Type / Target	Notes
control_id	`uuid`	Primary identifier; the control exists once across all regimes
control_name	`string`	Organisation-internal name
specification	`text`	What the control does, in operational terms
implementation	`text`	How it is implemented currently
objectives_addressed →	`1..n Control Objective Records`	Which objectives this control contributes to
regime_mappings	`structured`	How this control appears in each regime (CMMC ref, DEFSTAN ref, etc.)
effectiveness_observations →	`0..n observations`	Continuous evidence of operation
operational_owner	`Identity reference`	Who operates this control
effectiveness_owner	`Identity reference`	Who is accountable for its operation

ENTITY · **Taxonomy Record · versioned reference**

Attribute / Relation	Type / Target	Notes
taxonomy_id	`uuid`	Primary identifier
taxonomy_name	`string`	What this taxonomy classifies
version	`string`	Semantic version (changes are governed)
effective_period	`interval`	When this version is current
categories	`tree-structured`	The actual classification structure (hierarchical)
classification_rules →	`0..n Policy references`	Machine-evaluable rules for assigning categories
supersedes_taxonomy	`Taxonomy reference · nullable`	Previous version, if a revision
governance_authority	`Authority reference`	Who can revise this taxonomy

ENTITY · **Translation Rule Record**

Attribute / Relation	Type / Target	Notes
rule_id	`uuid`	Primary identifier
source_vocabulary	`string`	Vocabulary being translated from (e.g. ISO 31000)
target_vocabulary	`string`	Vocabulary being translated to (e.g. NIST RMF)
mapping	`structured`	The actual translation logic, with scope conditions
caveats	`text`	Where the mapping is partial or context-dependent
affected_entities	`set of entity types`	Which substrate entities this rule operates on
governance_authority	`Authority reference`	Who can change this rule
version	`string`	Semantic version

ENTITY · **Measurement Construct Record**

Attribute / Relation	Type / Target	Notes
construct_id	`uuid`	Primary identifier
construct_name	`string`	What is being measured
definition	`text`	Precise definition; ISO 27004 / NIST SP 800-55 form
unit_of_measure	`string`	Units; matters for cross-construct comparison
adequacy_threshold	`structured`	What value(s) constitute adequacy
data_sources	`structured`	Where the measurement data comes from
observed_against →	`0..n Control Records`	Which controls are measured this way
governance_authority	`Authority reference`	Who maintains this construct

ENTITY · **Substrate Change Record · audit**		
Attribute / Relation	**Type / Target**	**Notes**
change_id	`uuid`	Primary identifier; never modified after creation
changed_entity_id	`Entity reference`	What was changed
changed_entity_type	`string`	Which kind of entity
change_type	`enum`	Create, update, retire, restore
before_state	`structured · nullable`	Entity state before change (null for creates)
after_state	`structured · nullable`	Entity state after change (null for retirements)
changed_by	`Identity reference`	Who made the change
changed_at	`timestamp`	When
rationale	`text`	Why; required for substrate-content changes
authority_invoked	`Authority reference`	Under what authority the change was made

The Substrate Change Record is structurally non-negotiable. It is what makes second-order observation possible at all, because detecting drift in substrate content requires knowing what changed when, by whom, and on what basis. An implementation that treats substrate change auditing as a feature to be added later is an implementation that will have to be substantially rebuilt to support the framework's continuous-observation capabilities. The change record is foundational, not optional.

> **THE SUBSTRATE CHANGE RECORD**
>
> *Every change to substrate content — schemas, taxonomies, translation rules, control specifications, regime bindings — generates an immutable change record. This is what makes second-order observation technically possible. It is foundational, not optional, and an implementation that defers it will have to rebuild for it.*

Storage shapes for different entity types

The entities specified above do not all fit naturally in a single storage paradigm. Pretending they do produces awkward implementations; recognising the difference produces clean ones.

The chain entities — Regime Intent through Evidence Link — are graph-natural. Their relationships are first-class; queries against them are traversals ("what evidence links underwrite this regime's intent through what objectives?"); the cardinalities are many-to-many at the Control Objective layer; and the chain's value is precisely in being traversable from any end to any other. A graph database (Neo4j, Amazon Neptune, ArangoDB) handles these well. A relational database can handle them but at the cost of complex multi-table joins for what are conceptually simple queries.

The Control Record, Discipline Binding Record, and similar entities with structured attributes but limited deep relationships are relational-natural. They have well-defined attributes; their cardinalities are mostly one-to-many; their queries are predominantly filter-and-select; their transactional updates need ACID guarantees. PostgreSQL, SQL Server, MySQL handle these well.

The Taxonomy Record is hierarchical reference data with versioning, which fits neither paradigm cleanly. Taxonomies are trees (or sometimes DAGs); they evolve over time; queries against them often need point-in-time semantics ("what was the classification under version 2.3 of this taxonomy?"). Some teams handle this in a relational database with closure tables and effective-period columns; some use a document database with versioned tree documents; some use a graph database with type-of relationships. Each works; none is universally best.

The Substrate Change Record is append-only event-log natural. New change records are created continuously; existing records are never modified; queries against them are largely temporal ("what changed in this taxonomy between these dates?"). Event-sourcing patterns with Apache Kafka or a dedicated event store fit this well. Relational databases can implement append-only patterns through application-level constraints; the result works but is less efficient at scale.

The pragmatic answer for most implementations is a polyglot persistence approach — graph database for the chain, relational for operational entities, document or specialised storage for taxonomies, event store for the change log — connected through the event-driven plumbing the infrastructure piece described. The complexity is real but manageable; the alternative of forcing everything into one paradigm produces worse complexity in implementation and worse query performance in operation.

2 Lifecycle and governance of substrate content

Every entity in the substrate has a lifecycle. Some lifecycles are transactional — Evidence Link Records are created when evidence is generated, retired when evidence expires, otherwise stable. Some are editorial — Taxonomy Records evolve through deliberate revision over months or years. Some are governance-heavy — Control Objective Records change only with explicit organisational authority. The lifecycles differ, and the governance attached to each lifecycle differs accordingly.

Blast radius determines governance weight

A practical principle: the governance weight required for a change should scale with the change's blast radius — the number of other substrate elements the change affects. A change with small blast radius can be made with light governance; a change with large blast radius requires heavier governance.

Adding a new Evidence Link Record has very small blast radius. It affects one obligation, contributes to one or more regime evidence demonstrations, and produces no downstream effects on other substrate content. Light governance is appropriate: the operational role responsible for the obligation can add the link; an automated check verifies the artefact reference resolves; the change record captures who and when.

Revising a Taxonomy Record has very large blast radius. Every entity classified under the taxonomy is potentially affected; every policy rule that evaluates against the taxonomy needs review; every regime mapping that references taxonomy categories may need updating. Heavy governance is appropriate: explicit proposal, impact analysis, review by affected disciplines, staged rollout with parallel-run periods, explicit retirement of the prior version.

Changing a Translation Rule sits in between. The blast radius depends on how widely the rule is consumed — a translation between two rarely-used vocabularies has small blast radius; a translation between two widely-bound regimes has large blast radius. The governance has to adapt to the actual radius, which means the substrate has to make the radius visible.

> **BLAST RADIUS GOVERNANCE**
>
> *Governance weight scales with blast radius. The substrate has to make the radius visible — which entities depend on which, which regimes are affected by which taxonomies, which rules consume which schemas. Without visibility, governance is either too heavy everywhere (slowing everything) or too light everywhere (missing impacts).*

Versioning patterns by entity type

Reference entities need versioning. Operational entities mostly do not — they have lifecycle states (active, retired) but their identity is stable and they evolve only by being superseded. The versioning patterns differ accordingly.

Taxonomies and translation rules use semantic versioning with effective periods. Each version of the taxonomy has a definite start and end; queries can be made against historical

versions by specifying a point in time; the change records capture the transition from one version to the next; consumers can specify which version they are operating against if they need backward compatibility.

Control specifications use monotonic versioning without parallel versions. A control evolves; the latest version is current; historical versions are accessible for audit but not for current operation. Multiple versions in parallel produce more complexity than it is worth for this entity type.

Schemas use semantic versioning with deprecation notices. A schema can have multiple versions in use simultaneously during a deprecation period; consumers migrate from old to new; old versions are eventually retired. The substrate has to know which versions are in use by which consumers in order to know when deprecation is complete.

Regime intent and requirement records use external versioning — they reflect the regime's own versioning. CMMC 2.0 to CMMC 2.1 is a regime version change; the substrate captures both as related Regime Intent Records, with the older one retiring as the newer one takes effect. The substrate's versioning here mirrors the regime's, not the substrate's own internal version.

3 Identity, authority, and accountability

Most governance discussions conflate three things that PDCA+ treats as distinct: roles, authorities, and accountabilities. The distinction is consequential and the data model has to maintain it.

A role is a function a person performs. "Senior Configuration Manager," "Information Security Officer," "Risk Owner for Payment Systems" are roles. Roles describe what someone does. They are organisational positions, often documented in HR systems, often associated with job descriptions and reporting structures.

An authority is a decision a person can make. "Approve change requests for production payment systems," "Authorise risk acceptance for Tier 1 systems," "Sign off on regime retirement" are authorities. Authorities describe what decisions someone can commit the organisation to. They are bounded by scope, can be delegated, and require explicit grants.

An accountability is a consequence a person bears. "Accountable for the operation of authentication controls," "Accountable for the residual risk on payment processing" are accountabilities. Accountabilities describe what someone is on the hook for. They tend to be more durable than authorities — the person remains accountable even if their authority is delegated — and tend to attach at higher organisational levels.

These three are routinely conflated in GRC documentation and tooling. "The CISO owns information security" can mean any of: the CISO performs the role of CISO, the CISO has authority to make information-security decisions, the CISO is accountable for information-security outcomes. In practice it usually means the third with the second implied and the first taken for granted. The conflation works in conversation; it fails in substrate, because substrate has to be precise about which property applies to which entity.

The three entities, separately modelled

ENTITY · Identity Record

Attribute / Relation	Type / Target	Notes
identity_id	`uuid`	Primary identifier; corresponds to a real person or service
external_identity_ref	`string`	Reference to enterprise identity system (LDAP, IdP, etc.)
display_name	`string`	Human-readable name
identity_type	`enum`	Person, service account, group
status	`enum`	Active, suspended, terminated

ENTITY · Role Record

Attribute / Relation	Type / Target	Notes
role_id	`uuid`	Primary identifier
role_name	`string`	Organisational role title
role_description	`text`	What this role does
organisational_unit	`string`	Where this role sits
assigned_identities →	`0..n Identity Records`	Who currently holds this role

ENTITY · Authority Record

Attribute / Relation	Type / Target	Notes
authority_id	`uuid`	Primary identifier
authority_description	`text`	What decision this authority can make
scope_conditions	`structured`	Bounds on the authority (system class, value threshold, etc.)
granted_to_role →	`Role Record · nullable`	Whether this authority is role-attached
granted_to_identity →	`Identity Record · nullable`	Or directly to an individual
delegated_from	`Authority Record · nullable`	If this is a delegation, who originally held it
effective_period	`interval`	When this authority grant is active

ENTITY · Accountability Record

Attribute / Relation	Type / Target	Notes
accountability_id	`uuid`	Primary identifier
accountability_description	`text`	What this person/role is accountable for
held_by_role →	`Role Record · nullable`	Role-level accountability
held_by_identity →	`Identity Record · nullable`	Individual-level accountability
scope	`structured`	What the accountability covers
effective_period	`interval`	When this accountability is in force

In the chain entities specified earlier, references to authority and accountability appear repeatedly — the Control Objective Record has a governance_authority; the Obligation Record has a responsible_role; the Control Record has both operational_owner and effectiveness_owner. Each of these references one of these three entities. The data model makes the distinction explicit and the queries against it ("who has authority to authorise this risk acceptance?" "who is accountable for this control's effectiveness?") have precise answers.

Why the substrate change record references authority

The Substrate Change Record's authority_invoked attribute is what closes the loop. Every change to substrate content is made under some authority; the change record captures which authority was invoked; querying the change log tells the organisation not just what changed and when, but on what basis the change was made. This is what gives second-order observation its grip — the conjunction can observe whether the changes being made to substrate content are made under appropriate authority, surface anomalies (changes made without invocation, changes that exceed the scope of the invoked authority), and feed those observations back into the governance system.

> **THE AUTHORITY STRUCTURE, IN ONE SENTENCE**
>
> *Roles describe what people do; authorities describe what decisions they can make; accountabilities describe what they are on the hook for. The three are distinct and routinely conflated. The substrate keeps them separate, and every substrate change records the authority under which it was made, giving the conjunction the visibility it needs to observe governance itself.*

4 The path: staged adoption

With the data model and governance structure specified, the question of how an organisation actually gets there from where it is becomes addressable. The answer is staged — each stage adequate on its own terms, each stage providing the foundation for the next, each stage producing visible value so the organisation can sustain the journey across the multi-year timeframe a full implementation typically requires.

What follows is a five-stage path. The stages are not durations — "Stage 2" is not "year two." They are capability levels, and organisations move through them at different paces depending on size, regulatory exposure, existing infrastructure, and organisational appetite. The stages are designed so that an organisation can pause at any of the first four and operate adequately, while still being positioned to advance when the time comes.

Stage	What is in place	What is still required	What becomes possible
Stage 0	Compliance-check operating, GRC tool as artefact-of-record, fragmented data across disciplines, manual integration by GRC team	Everything below	Audit-cycle compliance, regime-by-regime; no structural reuse
Stage 1	Substrate data model designed, chain entities (Regime Intent through Evidence Link) implemented for one or two key regimes; one or two disciplines bound	Workflow engine, policy engine, event bus to fully replace manual integration	End-to-end traceability for bound regimes; control-objective reuse begins
Stage 2	Chain architecture live across multiple regimes; control inventory unified; translation rules between key regimes operational	Continuous observation infrastructure; second-order surfacing of substrate inadequacies	Genuine continuous compliance for bound regimes; per-audit reconstruction work largely eliminated
Stage 3	Second-order observation operational; substrate-change governance mature; identity-authority-accountability structure in production use	Strategic-compliance framing adopted at leadership level	Substrate self-monitors; regime-binding and retirement operations become routine
Stage 4	Strategic compliance practised; regime entry, exit, and risk-appetite shifts handled as substrate operations	Nothing structural; refinement only	Compliance posture as managed strategic variable; full scope of PDCA+ structural capabilities realised

What each stage gives, what each stage costs

Stage 0 is where most organisations live today. It works adequately for compliance-check operation in stable regime environments; it begins to fail when regimes multiply, flow-downs cascade, or regime change becomes frequent. The cost is structural: every audit cycle reconstructs evidence, every new regime adds parallel programme work, no learning compounds across cycles. The organisation can stay here indefinitely if its regime environment is simple enough; many organisations cannot.

Stage 1 is the first substantial investment. The substrate data model is designed; the chain entities are implemented; one or two regimes are addressed through the new structure. The

implementation effort is significant — typically several engineering-months at minimum, including data-model design, schema implementation, initial population of chain records, and integration with existing systems. The visible benefit is end-to-end traceability for the bound regimes: an auditor can trace evidence from a control back through the obligation, objective, requirement, to the intent. This is more than most organisations have today; the demonstration is what justifies progression to Stage 2.

Stage 2 is where the framework begins paying for itself operationally. The control inventory unifies; the translation rules between regimes make cross-regime evidence selection automatic; the integration plumbing eliminates large categories of manual GRC work. The investment now includes workflow engine, policy engine, and event bus integration — substantial but bounded. The benefit is the continuous-compliance mode the regime piece described: audit-cycle reconstruction work largely disappears, and the compliance function's time shifts to substrate stewardship.

Stage 3 brings the second-order observation capabilities online. The substrate begins observing its own observing apparatus; substrate-change governance matures into a deliberate practice; the identity-authority-accountability structure is in production use rather than aspirational. The investment shifts from infrastructure to instrumentation and governance: more observation logic, more pattern detection, more deliberate decisions about what to surface and to whom. The benefit is the framework becoming self-aware in the cybernetic sense — surfacing inadequacies in its own apparatus before they produce visible failures.

Stage 4 is the strategic capability. With everything else in place, the organisation can use the substrate as a strategic instrument: deliberate regime entry and exit, deliberate risk-appetite shifts, deliberate compliance posture reconfiguration in response to environmental change. The investment here is primarily organisational rather than technical — the technical infrastructure is mostly already built; what is required is the recognition and the leadership capability to use it strategically. Many organisations may take longer to reach Stage 4 than to reach Stage 3, because Stage 4 requires a kind of organisational maturity that is harder to engineer than infrastructure.

> **THE PATH, IN ONE SENTENCE**
>
> *Five stages, each adequate on its own terms, each laying foundations for the next. Most organisations are at Stage 0; reaching Stage 2 produces continuous compliance; reaching Stage 3 produces second-order observation; reaching Stage 4 produces strategic compliance. The technical work plateaus before Stage 4; the organisational work continues.*

What can go wrong on the path

Three failure modes are common enough to be worth naming.

Premature platform commitment. An organisation in Stage 0 selects a specific GRC platform or framework on the assumption that the platform will deliver everything through Stage 4. Most platforms are designed for Stage 0-1 operation; their architectures do not support Stage

2+ structural commitments. The platform investment then constrains the path. The remedy is to defer platform commitments until the substrate data model is clear and the architectural requirements for later stages are understood — which is what the technical-leg pieces in this series are for.

Stage skipping. An organisation tries to jump from Stage 0 directly to Stage 3, attracted by the visible benefits of continuous observation. The skipped stages turn out to have been load-bearing: without Stage 1's data model, Stage 3's observation has nothing structured to observe; without Stage 2's continuous integration, Stage 3's surfacing has nothing to detect inadequacies in. The remedy is patience and sequencing — each stage really does lay the foundation for the next, and the foundations cannot be implied.

Substrate ownership confusion. An organisation begins implementing the substrate but has not decided who owns it. The result is fragmentation: each discipline implements its own piece, with its own choices about schemas and taxonomies; the conjunction's coherence never emerges; the implementation produces what looks like Stage 2 but operates like Stage 0 with extra steps. The remedy is to establish substrate ownership explicitly before Stage 1 begins — typically a dedicated team or function, often within or alongside the GRC function, with clear accountability for substrate coherence across disciplines.

5 What this piece showed, and what comes next

This piece has shown the path: the substrate as a data model with specific entities and relationships; the lifecycle and governance that substrate content requires; the identity-authority-accountability structure that grounds the governance; and the staged adoption journey that takes an organisation from Stage 0 to Stage 4. The specifications are detailed enough that an architect can begin scoping a real implementation; they are abstract enough that the implementation choices remain with the implementing organisation.

What this piece has deliberately not done is solve the implementation. The DDL is not written; the specific tools are not selected; the integration patterns are not detailed; the resource model is not prescribed. These belong to the organisations doing the work. The framework's job is to specify what has to be true of the substrate, not to specify the substrate itself.

Three deliberate omissions remain — concerns that belong to the technical leg but not to this piece. The first is the operational concerns of running the substrate in production: observability, instrumentation, performance, scaling, failure modes. The second is the API surface that the substrate exposes — what other systems integrate against, what shapes those integrations take, what guarantees the substrate offers consumers. The third is code-as-compliance proper: how rules that evaluate against the substrate are written, tested, deployed, and governed as executable artefacts.

These three are not separate pieces. They cohere into the next two technical-leg pieces. The next piece — observing and integrating the substrate — takes the operational and API concerns together, because they are the same concern viewed from different angles: the substrate's production surface is what needs to be observed, and what is observed defines what integrations need to exist. The piece after that develops code-as-compliance with the

substrate, governance, and operational picture in place — at which point the methodology has somewhere to live, something to evaluate against, and a path to be deployed through.

The technical leg, then, is four pieces in sequence: infrastructure foundations (done), the substrate in detail (this piece), observing and integrating the substrate (next), and code-as-compliance (then). Each builds on its predecessors; each adds capability the previous lacked; each remains tool-agnostic in principle and implementable in practice. By the end of the leg, a technical architect should have what they need to build, operate, and govern a PDCA+ substrate in their own organisation.

> **WHAT THIS PIECE ESTABLISHED, IN ONE SENTENCE**
>
> *The substrate has a definite data model — entities, relationships, cardinalities — with content lifecycles that scale governance to blast radius, an identity-authority-accountability structure that keeps roles, authorities, and accountabilities distinct, and a five-stage adoption path that an organisation traverses at its own pace, with each stage adequate on its own terms.*