

THE GOVERNANCE-AS-CODE PLAYBOOK

Adopting FINOS AIGF for Agentic AI in Financial Services

A practical guide to policy definition, architecture, deterministic orchestration, and observability for agentic AI an FSI organization can actually run in production.

FINOS AI FUND

FINOS Fintech Open Source Foundation

The Governance-as-Code Playbook

Adopting FINOS AIGF for Agentic AI in Financial Services

First Edition

Draft — September 2026. Not for external distribution; contents subject to change ahead of first release.

Prepared as an internal guide on adopting FINOS's AI Governance Framework (AIGF) and its surrounding Governance-as-Code pipeline for agentic AI in financial services.

This guide references and draws on the open-source FINOS AI Governance Framework, CALM, and Fluxnova projects, the AIUC-1 certification standard, and the Linux Foundation's Tokenomics Foundation. Full sources are listed in the References chapter at the end of this guide.

For internal distribution.

Contents

Introduction 7
The Usual Pitfalls 7
Open Source Versus Proprietary: The Path This Guide Takes 8
Why This Gets an FSI Organization to ROI Faster 9
Chapter 1 — Governance-as-Code, in One Sentence 11
1.1 The Pipeline at a Glance 11
1.2 Pipeline Components and Tools 13
Chapter 2 — Where AIGF Fits in the Pipeline 15
2.1 Policy Definition — "What are the rules?" 15
2.2 Architecture — "How do we build it safely by default?" . 15
2.3 Controls — "How do we enforce it automatically?" . . . 15
2.4 Orchestration — "How does the agent actually run, within those controls?" 16
2.5 Observability and Feedback — "How do we know it's still working?" 16
Chapter 3 — The Determinism Problem, and Why Fluxnova Solves It 17
3.1 Why This Makes the Agentic Solution More Compliant, Not Just More Reliable 18
Chapter 4 — What This Means for Engineering Teams 19
4.1 CALM in Practice: Nodes, Relationships, and Controls . 19
4.2 Where Agentic Design Patterns Stop and Governance-as-Code Starts 21
4.3 What Changes Day to Day for Engineers 22
Chapter 5 — What This Means for GRC Teams 24
5.1 What Changes Day to Day for GRC 24
5.2 From Estimation to Production: GRC at Every Phase . . . 24
Chapter 6 — AIGF and AIUC-1: Differences and Complementarity 27
6.1 Two Different Questions, Two Different Layers 27
6.2 A Closer Look at AIGF 27
6.3 A Closer Look at AIUC-1 28
6.4 Side by Side 29
6.5 Using Them Together 29

6.6 Frequently Asked Questions . . . 30
Chapter 7 — A Step-by-Step Path Through the Pipeline . . . 32
7.1 Teams and Roles . . . 32
7.2 An Illustrative Timeline . . . 33
7.3 Iterative Delivery, Not a Cascade . . . 34
Step 1 — Classify the use case with the AIGF Use Cases Taxonomy. . . . 36
Step 2 — Run the heuristic assessment against the risk catalog. 36
Step 3 — Select the matching reference architecture and threat model. . . . 37
Step 4 — Translate the architecture into CALM specifications. 37
Step 5 — Draw the deterministic boundaries and hand them to Fluxnova. . . . 38
Step 6 — Embed compliance checks into CI/CD. . . . 38
Step 7 — Deploy with observability switched on from day one. . . . 38
Step 8 — Feed production evidence back into governance. . 39
Step 9 — Decide whether external certification is warranted. 39
Chapter 8 — Getting Teams Trained and Ready . . . 41
8.1 Training Is Role-Specific, Not a Single Curriculum . . . 41
8.2 Formal Training: Linux Foundation Education and Service Providers . . . 41
8.3 The First Use Case Is the Training Program . . . 42
8.4 Get a Seat in the Room Early . . . 43
8.5 The Failure Mode Specific to Training: Skew . . . 43
Chapter 9 — Codifying a Use Case . . . 45
9.1 Why This Step Is the Template . . . 45
9.2 Worked Example: An Agentic KYC Document Review Assistant . . . 45
9.3 Turning the Assessment into a Contribution . . . 47
9.4 Making the Pull Request . . . 48
9.5 Why This Is Worth the Effort . . . 48
Chapter 10 — Solution Architecture in Depth: From Reference Pattern to CALM Specification . . . 50
10.1 Selecting the Reference Architecture (Step 3) . . . 50
10.2 What the Pattern Leaves for This Use Case to Decide . . 51
10.3 Translating the Pattern into CALM (Step 4) . . . 51
10.4 Architecture Review and Sign-off . . . 55
10.5 From CALM to Fluxnova: What Carries Forward . . . 56
Chapter 11 — Fluxnova in Depth: Modeling the KYC Process . . . 57
11.1 From Mitigations to a Process Model . . . 57
11.2 Two AI-Native Fluxnova Constructs . . . 57
11.3 Anatomy of a Fluxnova Process Definition . . . 59
11.4 The KYC Process, End to End . . . 59
11.5 The DMN Decision Table . . . 61

11.6 Why This Satisfies the Determinism Requirement 61
11.7 From Process Model to CI/CD and Observability 62
Chapter 12 — Validating Controls in CI/CD: The KYC Build Gate 63
12.1 Why This Step Exists Between Building and Running . . 63
12.2 What Gets Checked, and Against What 63
12.3 A Worked Example: Testing the KYC Build 64
12.4 What Happens on Failure 64
12.5 From CI/CD Gate to Deployment 65
12.6 Runtime Conformance: A Control Plane for What's Deployed 65
Chapter 13 — Evals: Measuring What Determinism Doesn't Cover 69
13.1 What Determinism Doesn't Cover 69
13.2 Eval-Driven Development: The Methodology 69
13.3 What to Evaluate: Security and Functionality 70
13.4 Unit Evals vs. Regression Evals 70
13.5 A Worked Example: Evals for the KYC Extraction Tool . 71
13.6 Evals as a CI/CD Gate 72
13.7 From One-Time Gate to Continuous Signal 73
Chapter 14 — Observability in Depth: Evidence, Metrics, and the KYC Process in Production 74
14.1 Two Audiences, One Signal 74
14.2 The Three Signals: Traces, Metrics, and Logs 74
14.3 Task Executions: Success Rate, Duration, and Invocation Counts 75
14.4 Evidence of Control Usage: Proving the Mitigations Held 76
14.5 Dashboards for Different Audiences 77
14.6 From Observability to the Feedback Loop 77
14.7 Generating Compliance Reports: Per-Case and Aggregate 78
Chapter 15 — Tokenomics: Attribution, Cost, and Value 81
15.1 Tokenomics Is an Attribution Problem, Not a Metering Problem 81
15.2 Three Axes of Attribution: Agent, Use Case, Department 82
15.3 Tagging Consumption at the Source 82
15.4 Beyond Tokens: The Full Cost of AI 83
15.5 From Consumption to Value: The Net Benefit Equation 84
15.6 A Worked Example: KYC Tokenomics 85
15.7 Governing Cost the Same Way We Govern Risk 86
15.8 Building Value Accountability into the Pipeline 87
15.9 A Control Plane Screen for Tokenomics 88
15.10 One Control Plane, Four Views 89
Conclusion — Why This Matters More Than Another Framework 91
What This Guide Covered 91
The Benefits, Taken Together 93
References 95
FINOS AI Governance Framework (AIGF) 95

Governance-as-Code, the FINOS AI Fund, and Common Cloud Controls 95
CALM (Common Architecture Language Model) 95
Fluxnova 96
AIUC-1 96
Tokenomics 97
Evals 97

Introduction

A note on terms before we start: this guide uses "FSI" and "FSI organization" to mean any financial services institution — a bank, but equally an insurer, an asset manager, a payments provider, or another regulated financial firm. The pipeline and every example in this guide were built around a banking use case, because that's where AIGF and Fluxnova have the deepest early adoption, but nothing about the argument is bank-specific. Wherever this guide says "an FSI organization," read it as applying to whichever kind of regulated financial institution you actually work for.

Ask an FSI organization's compliance team how they govern a new AI system today, and you'll usually hear some version of the same story: a policy document, a risk assessment spreadsheet, a review meeting, a sign-off. Ask the engineering team building that system, and you'll hear the mirror image: governance shows up late, as a gate at the end of the pipeline, disconnected from the code and largely manual.

That gap — governance as paperwork versus engineering as pipeline — is exactly what FINOS's Governance-as-Code initiative, backed by the new FINOS AI Fund, is built to close. And the FINOS AI Governance Framework (AIGF) isn't a bystander to that effort. It's the foundation the whole pipeline is built on.

The Usual Pitfalls

Before getting into how the pipeline works, it's worth naming what tends to go wrong when FSI organizations build agentic AI without something like it in place. None of these are hypothetical — they're the recurring patterns behind delayed launches, failed audits, and agents quietly pulled from production:

- **Governance bolted on at the end.** A team builds an agent, gets it working, and only then brings it to risk and compliance for review — by which point the architecture is set, the autonomy boundaries are implicit in the code rather than designed in, and any serious finding means a costly redesign rather than a small adjustment.
- **Every team reinvents its own risk assessment.** Without a shared taxonomy, one team's "high risk" is another team's "minor internal tool," and there is no consistent basis for the board or a regulator to compare use cases across the institution — or even to trust that every use case was assessed with the same rigor.
- **Shadow AI and ungoverned proliferation.** Business units and individual teams adopt agentic tools faster than central governance can track them, because the sanctioned path is slow and manual — leav-

ing the institution with agents in production that nobody centrally assessed.

- **Non-determinism quietly accepted, then discovered the hard way.** An agent behaves acceptably in a proof of concept, where a wrong answer costs nothing, and the same non-deterministic behavior ships into a regulated workflow where a skipped control point becomes a reportable incident — because nobody drew a hard line between what the model can decide freely and what must be fixed and enforced.
- **No reconstructable audit trail.** When a regulator or an internal auditor asks how a specific automated decision was reached, the honest answer is often "the model decided," which satisfies nobody — the reasoning path wasn't fixed, so the evidence trail wasn't fixed either, and reconstructing it after the fact is expensive or impossible.
- **Compliance treated as a one-time gate, not a continuous state.** A system passes its launch review and is then left unmonitored for drift — the model, the data, or the threat landscape moves on, and nobody notices until something goes wrong, because the governance process had no mechanism for continuous feedback.
- **Duplicated effort across the industry.** Every FSI organization building, say, a KYC or AML agent independently re-derives the same risk categories and mitigations that dozens of peer institutions have already worked out — burning months of legal, risk, and engineering time on a problem that has effectively already been solved elsewhere.
- **Poorly defined escalation and human oversight.** "Human in the loop" gets asserted in a policy document without being enforced anywhere in the system, so in practice the agent's autonomy in production is broader than anyone signed off on.

Open Source Versus Proprietary: The Path This Guide Takes

There are two broad ways an FSI organization can approach governance for agentic AI, and it's worth being explicit about which one this guide follows.

The first is to buy a proprietary governance platform from a large vendor. This can get a program running quickly, but it comes with real trade-offs: the risk logic and control mappings typically live inside a closed, licensed product that the FSI organization cannot fully inspect, audit, or adapt to its own specific regulatory obligations; the FSI organization's governance program becomes dependent on that vendor's roadmap and pricing; and the investment the FSI organization makes in configuring the platform for its use cases doesn't transfer if the relationship ends, and doesn't benefit any other institution facing the same problem.

The second path — the one this guide is built around — is the open-source route: FINOS's AI Governance Framework and the surrounding Governance-as-Code pipeline (CALM, Common Cloud Controls, Fluxnova, and the observability layer), developed collaboratively by the financial institutions and technology providers that actually have to answer to regulators. This isn't a philosophical preference. It carries concrete, practical advantages for an FSI organization building its own agentic AI:

- **Transparency.** The risk catalog, the mitigations, and the taxonomy are all open text a regulator, an auditor, or an internal risk committee can read directly — nothing is a black box the FSI organization has to take on faith from a vendor.
- **No lock-in.** Because the framework is open source and vendor-agnostic, the FSI organization's governance program isn't hostage to one company's licensing terms, roadmap, or continued existence.
- **Shared cost, compounding benefit.** Every risk category, mitigation, and reference architecture that AIGF already covers is work the FSI organization does not have to redo — and, as Chapter 9 shows in detail, work the FSI organization contributes back (like a KYC use case) becomes reusable infrastructure for every other institution using the framework, the same way the FSI organization benefits from what Morgan Stanley, Citi, NatWest, and others have already contributed.
- **Control stays with the institution.** The FSI organization can extend, fork, and adapt the framework to its own regulatory footprint, rather than waiting on a vendor to prioritize a feature request.

Why This Gets an FSI Organization to ROI Faster

The pitfalls above aren't just governance problems — they're expensive ones, in time and in money. Building on AIGF's Governance-as-Code pipeline instead of starting from a blank page, or from a proprietary black box, shortens the path to a return on investment in several concrete ways:

- **Less rework, because governance is designed in rather than bolted on.** Catching a risk during architecture selection (Chapter 2) costs a design decision; catching the same risk after launch costs a rebuild. Fewer late-stage redesigns means agents reach production faster.
- **No proprietary licensing overhead.** Building on an open-source framework avoids the recurring per-seat or per-agent costs of a closed governance platform, and avoids the switching costs that come with vendor lock-in.
- **Audit and certification evidence is a byproduct, not a project.**

Because the pipeline generates execution traces, CI/CD validation history, and risk mappings automatically as agents are built and run, both internal audits and external certification (Chapter 6) start from existing evidence rather than a costly, from-scratch evidence-gathering exercise.

- **Reuse compounds across the agent portfolio.** The first use case an FSI organization runs through this pipeline is the slow one — Chapter 9 shows what that first pass looks like in full. The second, and the fiftieth, move faster, because the taxonomy, the risk catalog, and the reference architectures are already in place and already proven.
- **Faster regulatory engagement.** Because AIGF's risk catalog is already mapped to frameworks regulators recognize — the EU AI Act, FCA Consumer Duty, PRA operational resilience — conversations with supervisors start from shared vocabulary rather than a bespoke internal framework nobody outside the FSI organization has seen before.

The chapters that follow walk through exactly how this works in practice — starting with what the pipeline is, then how each stage fits together, then a full worked example that turns a single use case into a governed, running, and auditable agent.

Chapter 1 — Governance-as-Code, in One Sentence

Agentic AI systems — agents calling tools, invoking MCP servers, making autonomous decisions — move and change far faster than traditional manual, siloed governance can track. The raw material an FSI organization actually starts from is fragmented: regulations, industry guidelines, technical standards, and best practices scattered across dozens of sources, none of them machine-readable and none of them talking to each other. FINOS's answer is to turn that fragmented landscape into something machine-readable and runtime-observable: an end-to-end pipeline covering policy definition, architecture, controls, orchestration, and observability, feeding back on itself as a continuous loop. Instead of governance being a document engineers consult and a checklist compliance signs off on, it becomes something that runs alongside the code — checked automatically, continuously, and with evidence generated as a byproduct rather than a separate task.

That pipeline is assembled from several open-source FINOS projects working together. AIGF sits at the center as the thing that defines *what* to govern, while the surrounding tools handle *how* that gets enforced and observed.

1.1 The Pipeline at a Glance

Before going stage by stage, it helps to see the whole shape at once. Fragmented inputs feed into five connected stages, with a feedback loop closing back to policy definition, and an optional branch out to external certification once internal governance is mature:

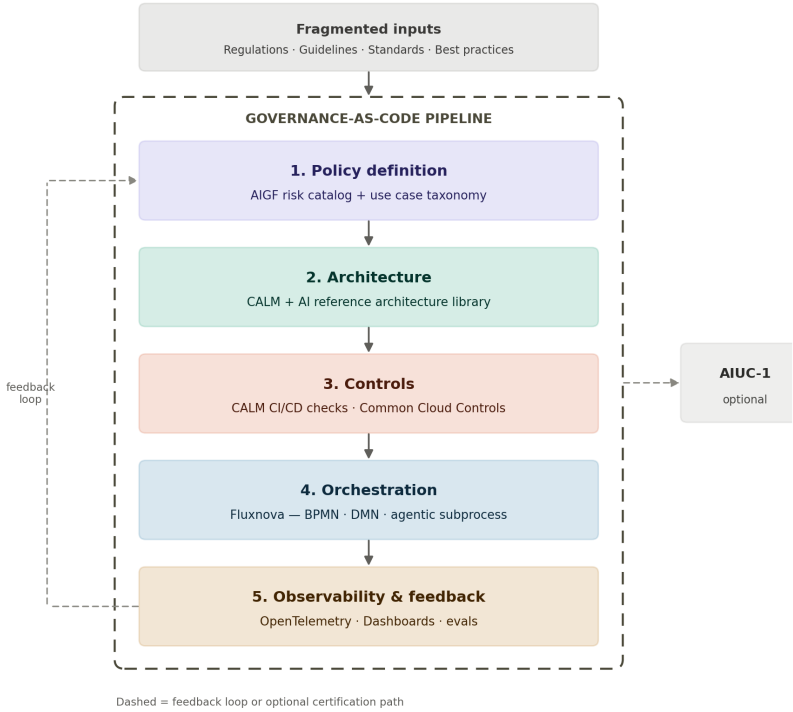


Figure 1. The Governance-as-Code pipeline as a bordered whole: fragmented inputs feed the pipeline, five stages run inside it with a continuous feedback loop, and AIUC-1 certification attaches to the pipeline itself rather than to any single stage.

Reading the diagram: the dashed border is the important addition here, and it's deliberate — it marks the Governance-as-Code pipeline as a single, bounded whole, not a loose sequence of independent stages. Two things attach to that border rather than to any one stage inside it. At the top, fragmented inputs — regulations, industry guidelines, technical standards, and internal best practices, none of them arriving in a form a pipeline can consume directly — feed into the pipeline as a whole; policy definition is where that synthesis actually happens, but the raw material is something the entire pipeline exists to digest, not something Step 1 alone consumes in isolation. On the right, AIUC-1 attaches to the same border rather than branching off orchestration or any other single stage, because certification isn't an audit of one component — it's an assessment of what the governed pipeline, taken together, actually produces. Inside the border, the five stages run in their usual order: policy definition's risk catalog and tax-

onomy flow into architecture's CALM specifications, which split into two distinct concerns — controls, where CI/CD checks and Common Cloud Controls enforce policy automatically against every change, and orchestration, where Fluxnova actually executes the agentic process within the boundaries those controls require. The running system is then watched continuously in observability and feedback, and what's learned there flows back up into policy definition, closing the loop entirely inside the pipeline's own border. AIUC-1 remains outside that loop by design — a deliberate, optional step an FSI organization takes once the internal pipeline has already produced the evidence to support it, covered in full in Chapter 6.

1.2 Pipeline Components and Tools

The pipeline isn't one product — it's a set of open-source FINOS projects, each responsible for one part of the flow. The table below lays out every component referenced in this guide, which stage it belongs to, and what it actually does.

Stage	Component / tool	Role in the pipeline
Policy definition	**AIGF Risk & Mitigation Catalog**	The core set of financial-services-specific AI and agentic risks, each paired with concrete mitigations, mapped to frameworks like NIST, OWASP, and the EU AI Act.
Policy definition	**AIGF Use Cases Taxonomy**	Classifies AI systems by type, architecture pattern, and data handling needs, so risk and controls scale to what a system actually does.
Architecture	**AI Reference Architecture Library**	Pre-vetted, composable architecture patterns (single agent, multi-agent, tool-using agents) with security and oversight boundaries designed in.
Architecture	**CALM (Common Architecture Language Model)**	Expresses reference architectures and AIGF policy as machine-readable, executable specifications — the bridge between regulatory language and actual code.
Controls	**CALM CI/CD integration**	Embeds compliance checks directly into build pipelines, continuously validating agents and orchestration logic against AIGF-defined policy.
Controls	**Common Cloud Controls (CCC)**	Extends governance into the infrastructure layer, mapping regulatory expectations to actionable technical controls for cloud and hybrid AI workloads.

Stage	Component / tool	Role in the pipeline
Orchestration	**Fluxnova**	BPMN/DMN-based orchestration engine that pins down deterministic process boundaries and decision logic around non-deterministic agent behavior, with full execution tracing.
Observability & feedback	**OpenTelemetry**	Open instrumentation standard capturing real-time traces, metrics, and logs from running agents.
Observability & feedback	**Dashboards (e.g. Grafana)**	Visualizes observability data — behavior, cost, execution traces — as live operational and compliance signal.
Observability & feedback	**Evals**	Continuous evaluation of agent behavior in production, feeding results back into the risk profile and, where relevant, back into AIGF itself.
Cross-cutting	**FINOS AIGF MCP Server**	Turns requirements documents and architectural descriptions into governed, in-workflow guidance by connecting an AI agent directly to the taxonomy and risk catalog.
Cross-cutting	**FINOS AI Fund**	The FSI-led funding and strategic-direction body (launched June 2026) backing AIGF and the rest of the Governance-as-Code pipeline.
External (optional)	**AIUC-1**	An independent, insured, adversarially-tested certification an FSI organization can pursue for specific high-risk or customer-facing agents once internal governance is in place — see Chapter 6.

A few things worth noting about this list. First, nothing here is proprietary — every component is open source, governed collaboratively by FINOS member institutions, which is the central theme of this guide (see the Introduction). Second, the components are additive, not exclusive: an FSI organization doesn't have to adopt all of them on day one to get value — starting with the AIGF risk catalog and taxonomy alone (Chapter 9 walks through exactly how) already produces a usable governance program, with CALM, CCC, Fluxnova, and the observability layer added as the engineering pipeline matures. Third, AIUC-1 is listed as external and optional deliberately — it's not part of the Governance-as-Code pipeline itself, but a certification an FSI organization can layer on top of it, as Chapter 6 explains in detail.

Chapter 2 — Where AIGF Fits in the Pipeline

Think of the pipeline as five stages fed by one fragmented starting point, each answering a different question.

2.1 Policy Definition — "What are the rules?"

This is where the fragmented inputs from the top of Figure 1 actually get synthesized. No institution starts with a clean slate — it starts with regulations (the EU AI Act, evolving supervisory guidance), industry guidelines, technical standards, and internal best practices, scattered across sources that don't reference each other and don't map cleanly onto engineering work. AIGF's home turf is turning that fragmentation into one structured, standardized artifact: its risk and mitigation catalog gives financial institutions a way to define governance controls for AI and agentic systems — covering model selection, agent autonomy levels, operational oversight, and supply chain integrity — with each entry mapped back to the regulatory expectations it actually satisfies. Crucially, AIGF expresses these as machine-readable compliance specifications, not prose — which is what makes everything downstream possible. Alongside the risk catalog, AIGF's Use Cases Taxonomy classifies AI systems by type, architecture pattern, and data handling needs, so a fraud-detection model, an autonomous trading agent, and an internal coding assistant aren't governed by the same blunt checklist — each gets risks and controls appropriate to what it actually does.

2.2 Architecture — "How do we build it safely by default?"

Building on that taxonomy, FINOS's AI Reference Architecture Library — powered by CALM (Common Architecture Language Model) — turns AIGF's catalog into concrete, adoptable reference architectures and threat models for patterns like multi-agent systems and tool-using agents. CALM's role is to transform static governance policy into machine-readable, executable specifications, bridging high-level regulatory requirements and actual technical implementation, expressed in a structured, code-like format.

2.3 Controls — "How do we enforce it automatically?"

This is where governance actually becomes code. CALM allows institutions to embed compliance checks directly into CI/CD pipelines and development workflows, continuously validating AI models, agents, and orchestration logic against AIGF-defined policies — before deployment and throughout the system's life, not just at a one-time review gate. Common Cloud Controls (CCC) extends this into the infrastructure layer, mapping

regulatory expectations to actionable technical controls for AI workloads running in cloud and hybrid environments. Controls are deliberately a separate concern from what actually runs the agent: this stage answers whether a given change is allowed, not how the agent's process actually executes — that's the next stage's job.

2.4 Orchestration — "How does the agent actually run, within those controls?"

If controls define what's allowed, orchestration is where an agentic process actually executes inside those boundaries. Fluxnova operationalizes this at the workflow level, letting institutions define and automate agentic processes — multi-step decisions, tool invocations, human-in-the-loop checkpoints — using BPMN/DMN-based orchestration that's governance-embedded by construction, not bolted on afterward. This is also the stage Chapter 3 exists to justify: an LLM is non-deterministic, but the process wrapped around it doesn't have to be, and Fluxnova is the mechanism that keeps that distinction real at runtime rather than aspirational.

Orchestration isn't limited to agents a single team built and hosts itself. Fluxnova can just as readily invoke an external agent — one belonging to another business unit inside the same institution, or one supplied by a partner or vendor outside it — as a step within the workflow, the same way it invokes any other tool. What makes this safe rather than a governance blind spot is that the external agent never operates outside the process: the workflow supporting the use case still owns the sequencing, still gates the step behind whatever deterministic boundary applies, and still traces the call the same way it traces an internal one. An external agent is a participant the process controls, not a hole in the boundary the process draws around itself.

2.5 Observability and Feedback — "How do we know it's still working?"

The pipeline closes the loop with real-time monitoring, cost tracking, and continuous improvement through evals and operational data — built on open tooling like OpenTelemetry for instrumentation and dashboards (Grafana is a common choice) for visualization — so governance isn't a point-in-time judgment but a live signal that feeds back into policy and architecture as agents evolve in production.

Chapter 3 — The Determinism Problem, and Why Fluxnova Solves It

Here's the tension at the heart of every agentic AI compliance conversation: LLMs are non-deterministic by nature. Ask the same question twice and you can get two different answers, two different tool-call sequences, two different paths through a task. That's a feature for creativity and adaptability — and a serious liability for an FSI organization, where a regulator, an auditor, or a court needs to know that a specific process was followed the same way every time, that specific approval gates were never skipped, and that the sequence of decisions leading to an outcome can be reconstructed after the fact.

An agent that "decides for itself" whether to check a sanctions list, escalate to a human, or proceed autonomously isn't just unpredictable — from a governance standpoint, it's unauditable. If the reasoning path isn't fixed, the evidence trail isn't fixed either, and an FSI organization cannot certify or attest to a process it can't reliably reproduce.

Fluxnova's role in the pipeline is to draw a hard line between the parts of a workflow that must be deterministic and the parts where an LLM's judgment genuinely adds value — and to keep the deterministic parts deterministic even when AI is involved throughout. Built on BPMN and DMN, with a focus on audit-ready execution, Fluxnova gives financial institutions the same operational guarantees traditional distributed systems have always required, with probabilistic decision-making layered on top only where it's supposed to be. Concretely:

- **The process backbone is fixed, modeled, and enforced.** A workflow — say, a loan-servicing exception or a KYC remediation case — is expressed as a BPMN model: a defined sequence of steps, gateways, and approval points that the orchestration engine executes exactly as specified, every time. The LLM doesn't get to skip a mandatory human-in-the-loop checkpoint or reorder a sequence of controls because it decided that was more efficient — the process model is the authority, not the model's own judgment about what to do next.
- **DMN pins down the decision logic that must be consistent.** Wherever a decision needs to be rule-based and reproducible — eligibility thresholds, escalation criteria, risk scoring bands — Decision Model and Notation captures that logic explicitly and separately from the LLM, so the same inputs always produce the same decision, and that decision logic can be reviewed, versioned, and audited independently of any model's behavior.
- **The LLM operates inside guardrails, not instead of them.** Non-deterministic reasoning gets scoped to where it belongs — summa-

rizing a document, drafting a response, interpreting ambiguous free-text input — while the orchestration layer around it stays fixed.

- **Every execution is traced.** Because the engine executes and traces workflows as a matter of course, business and compliance teams get full visibility into exactly where a given case is, what path it took to get there, and whether service-level and control commitments were met — turning "trust us, the agent behaved" into a reconstructable execution record.

3.1 Why This Makes the Agentic Solution More Compliant, Not Just More Reliable

- **Reproducibility becomes provable, not asserted.** When a regulator asks how a specific customer outcome was reached, the answer isn't "the model decided" — it's the exact BPMN path and DMN decision the case followed.
- **Control points can't be silently bypassed.** Mandatory checks — sanctions screening, four-eyes approval, escalation thresholds — live in the process model itself, not in a prompt instructing the model to "remember" to do them. A prompt is a request; a BPMN gateway is an enforcement point.
- **Non-determinism gets contained, not eliminated.** FSI organizations don't need to strip agentic AI of its adaptive value to make it compliant — they need to make sure the unpredictable part of the system can't touch the part that has to be predictable.
- **Evidence generation is automatic.** Because execution is traced by default, the audit trail an examiner or an AIUC-1-style adversarial tester would ask for already exists as a byproduct of running the process, rather than needing to be reconstructed or, worse, self-reported.

This is also why Fluxnova sits naturally inside AIGF's broader risk catalog rather than beside it: AIGF defines which risks demand a deterministic control point — an autonomy boundary, an oversight gate, a data-handling rule — and Fluxnova is the mechanism that actually enforces that boundary at runtime, every single execution, regardless of what the underlying model decides to do.

Chapter 4 — What This Means for Engineering Teams

For engineers, the practical shift is significant: governance stops being someone else's job that shows up at the end. This chapter looks at what that shift actually looks like in the artifacts engineers write and review day to day, starting with CALM itself.

4.1 CALM in Practice: Nodes, Relationships, and Controls

CALM is a JSON-based, version-controlled schema — the kind of file that sits in a repository and gets reviewed in a pull request like any other. Its three core building blocks are nodes (the components of an architecture), relationships (how those components connect), and controls (the requirements attached to a node or relationship, and how they're satisfied). The point of CALM isn't specific to any one use case, so the smallest useful illustration is a "hello world" agent rather than a full financial-services scenario — the mechanics below are exactly what a real use case builds on.

A minimal CALM fragment starts with the nodes and the relationships between them: a simple greeter agent, one tool it can call, and a place it logs its output.

```
{
  "nodes": [
    {
      "unique-id": "hello-world-agent",
      "node-type": "service",
      "name": "Hello World Agent (agentic ad-hoc subprocess)",
      "description": "LLM-driven subprocess bounded to a single inner tool:
  composing a greeting",
      "interfaces": []
    },
    {
      "unique-id": "greeting-tool",
      "node-type": "service",
      "name": "Greeting Composer Tool",
      "description": "Formats a greeting string for a given name and language"
    },
    {
      "unique-id": "interaction-log",
      "node-type": "system",
      "name": "Interaction Log",
      "description": "Receives the agent's output for auditing"
    }
  ],
```

```
  "relationships": [
    {
      "unique-id": "agent-calls-greeting-tool",
      "connects": { "source": "hello-world-agent", "destination":
  "greeting-tool" },
      "description": "Agent invokes greeting composition as a bounded tool
  call"
    },
    {
      "unique-id": "agent-writes-log",
      "connects": { "source": "hello-world-agent", "destination":
  "interaction-log" },
      "description": "Agent's output is written to the interaction log"
    }
  ]
}
```

Notice what this fragment does and doesn't say. It says the agentic subprocess exists, what it's called, and exactly which one tool and which downstream system it's allowed to talk to — the relationships list is the complete set of connections; there is no relationship to anything else, which means a reviewer (or an automated CI check) can see the agent's entire reachable surface at a glance. It doesn't yet say anything about compliance requirements — that's what controls are for.

Controls attach directly to a node, pairing a requirement (what must be true) with a configuration (the specific values that satisfy it for this case). Here, a simple content-safety control on the same agent:

```
{
  "nodes": [
    {
      "unique-id": "hello-world-agent",
      "node-type": "service",
      "name": "Hello World Agent (agentic ad-hoc subprocess)",
      "controls": {
        "content-safety": {
          "description": "Controls governing what the agent is permitted to
  output",
          "requirements": [
            {
              "control-requirement-url": "https://example.org/controls/schema/
  no-restricted-languages.json",
              "control-config-url": "https://example.org/controls/config/hello
  -world-allowed-languages.json"
            }
```

```
          ]
        }
      }
    }
  ]
}
```

The requirement file ("no-restricted-languages.json") is the reusable, versioned statement of the rule itself — in this case, that the agent may only respond in a language from an approved list. The configuration file is what makes that rule concrete for this specific node: the actual list of approved languages. A real use case follows exactly this same pattern, just with requirements that matter more than a greeting's language — a control stating that a sanctions match can never be auto-cleared, say, looks structurally identical to the one above, with a different requirement and a different configuration behind it. Later chapters build one of those in full.

Because this is JSON, checking it is mechanical. A CI job can confirm that every node tagged as an agentic subprocess has a non-empty controls block, that every relationship listed matches a tool the agent is actually configured to call, and that no new relationship was added without a matching control review — the same category of check a security team already runs against infrastructure-as-code today, just pointed at the architecture of the agent instead of the cloud resources around it.

4.2 Where Agentic Design Patterns Stop and Governance-as-Code Starts

Most of what engineering teams read about building agents right now comes from design-pattern guidance — Anthropic's widely-read write-up on building effective agents, for instance, lays out patterns like prompt chaining, routing, parallelization, orchestrator-workers, and evaluator-optimizer, alongside the general shape of an autonomous agent loop. Other common patterns in circulation — ReAct-style reasoning-and-acting loops, plan-and-execute — cover similar ground. These are genuinely useful, and nothing in this guide argues against using them.

But it's worth being precise about what these patterns are for. Every one of them answers a capability question: how should an agent structure its reasoning and tool use to complete a task well, reliably, and efficiently. None of them, on their own, answers a compliance question: which of this agent's actions require a human sign-off that cannot be skipped, how do you prove after the fact that a specific decision was reached the same way it would be reached again, or how does this pattern map to the risk categories

a regulator or an internal audit committee actually cares about. That's not a criticism of the patterns — it's simply outside what they set out to do.

- **Orchestrator-workers**, for example, is a pattern for decomposing a task across multiple specialized calls — it says nothing about which of those worker calls, if any, needs a mandatory human checkpoint before its output can be acted on.
- **Routing** decides which specialized path handles a given input — it doesn't specify what evidence needs to exist afterward to show a regulator which path a specific case took and why.
- **Evaluator-optimizer** improves output quality through iterative self-critique — it has no concept of a control that can never be overridden regardless of how confident the evaluator becomes.

This is exactly the gap Governance-as-Code fills, and it does so by sitting one level below the pattern rather than replacing it. An engineering team is free to implement whatever reasoning pattern best suits the tools available inside an agentic ad-hoc subprocess like the one above — a simple routing decision between tools, a small ReAct-style loop, or nothing more sophisticated than a fixed call sequence. Whichever pattern is chosen lives entirely inside the subprocess boundary defined in the CALM fragment. The pattern determines how well the agent reasons within that boundary; the CALM controls, the AIGF risk mapping, and the Fluxnova process definition determine that the boundary itself cannot be crossed, that every action is traced, and that the specific decision point a regulator will ask about has an answer that doesn't depend on trusting the pattern's judgment. Put another way: design patterns make the agent good at its job inside the box. Governance-as-code is what defines and enforces the box, and hands you the evidence that it held.

4.3 What Changes Day to Day for Engineers

- **Compliance checks run where code already runs.** Because AIGF policies get expressed through CALM as executable specifications, they slot into existing CI/CD pipelines. A pull request that changes an agent's tool permissions or data access can be checked against AIGF's risk catalog automatically, the same way a linter or test suite runs today.
- **Reference architectures remove guesswork.** Rather than each team inventing its own pattern for a multi-agent system and hoping it satisfies risk and compliance later, the AI Reference Architecture Library gives engineers pre-vetted, composable patterns with security, observability, and cost boundaries already designed in.
- **Governance becomes a build-time signal, not a launch blocker.**

Automated policy validation surfaces issues while a system is being built, not in a review meeting three weeks before a planned launch — which means fewer late-stage redesigns and a much shorter path from prototype to production.

- **The FINOS AIGF MCP Server makes this conversational and contextual.** It takes requirements documents and architectural descriptions and turns them into actionable, governed insights — flagging relevant risks and mitigations directly inside the development workflow, with AI-generated outputs quality-assured by human review before anything is acted on.

Chapter 5 — What This Means for GRC Teams

For risk, compliance, and model governance functions, the shift is just as significant, though it looks different from their side of the table.

5.1 What Changes Day to Day for GRC

- **Consistency at scale.** The Use Cases Taxonomy means every AI system in the institution gets classified and governed the same way, regardless of which team built it.
- **Evidence, not attestations.** Because controls run automatically in the pipeline and observability data is captured continuously, GRC teams get generated evidence of compliance rather than relying on engineers to self-report that they followed the process.
- **Faster, better-informed reviews.** When a system reaches a formal risk review, the reference architecture, the AIGF risk mapping, and the CI/CD validation history already exist.
- **Direct line to regulatory frameworks.** Because AIGF's catalog is explicitly mapped to regulations like the EU AI Act, GRC teams aren't translating internal controls into regulatory language after the fact.
- **A seat at the table industry-wide.** AIGF is being shaped by the same institutions that have to answer to the same regulators — Morgan Stanley, NatWest, DTCC, RBC, and others are founding backers of the FINOS AI Fund driving this work.

5.2 From Estimation to Production: GRC at Every Phase

The pitfalls in the Introduction included governance bolted on at the end and compliance treated as a one-time gate. Both come from the same root cause: in a traditional model, GRC's involvement is concentrated into a single late-stage review, because that's the only point where there's something concrete enough to review. Governance-as-code spreads that involvement across the entire lifecycle instead, in a form appropriate to each phase — which changes not just when GRC shows up, but how much effort each appearance takes. Walking the same lifecycle a KYC-style use case actually moves through:

- **Estimation and scoping.** Traditionally, compliance review time is a guess — a project is sized without really knowing how much scrutiny it will need, because that depends on details nobody has worked out yet. With the taxonomy in place, classification (Step 1) can happen the moment a use case is proposed, before a line of code is written. A use case that classifies as high-autonomy, high-data-sensitivity gets

flagged for a heavier review budget at the estimation stage itself; a low-autonomy internal tool gets sized accordingly. GRC's role here is defining and maintaining the taxonomy's categories, not reviewing individual projects one at a time.

- **Design.** This is where the risk shortlist and mitigation mapping (Step 2) and the reference architecture selection (Step 3) happen. GRC's involvement is direct but bounded: reviewing a specific, structured artifact — the kind produced in Chapter 9 — rather than an open-ended conversation about a system that doesn't exist yet. Because the taxonomy and risk catalog already exist, this review is comparing a proposal against a known list, not inventing the list from scratch for this project.
- **Development.** Once CALM specifications and Fluxnova process definitions are being written (Steps 4 and 5), GRC's role shifts from reviewing artifacts by hand to defining the automated checks that will review every subsequent change — the CI/CD policy checks from Section 4.1 and Chapter 2. GRC does not read every pull request; GRC ensures the control requirements those pull requests get checked against are correct and current.
- **Testing and pre-production.** With the process modeled in Fluxnova (Chapter 11), the DMN decision tables can be unit-tested directly against the outcomes compliance actually signed off on — exactly as Section 11.5 describes. This is also the point at which a decision on AIUC-1 certification (Step 9) typically gets made for higher-risk use cases, since the artifacts an external audit would need are now largely in place.
- **Deployment.** CI/CD policy checks passing becomes a release gate (Step 6) rather than a manual sign-off email. A system cannot reach production with a failing control check any more than it could ship with a failing test suite — which means the deployment-time GRC conversation is about exceptions, not about running the checklist from the top for every release.
- **Production and monitoring.** This is the biggest structural change from the traditional model. Instead of a periodic audit reconstructing what happened from logs after the fact, GRC has a live view through the observability layer (Step 7) — dashboards showing case volumes by outcome, escalation rates, drift signals from evals. A compliance question about the KYC agent's behavior last quarter is answered by querying existing data, not by requesting a special audit.
- **Continuous improvement.** When production data surfaces something the original risk assessment didn't anticipate — an unexpected escalation pattern, a drift in extraction confidence — that finding

feeds back into the risk catalog (Step 8), the same feedback loop shown in Chapter 1's diagram. GRC's role here is deciding whether the finding changes the mitigation for this use case, or whether it's significant enough to propose back into AIGF itself, extending the benefit to every other institution using the framework, exactly as Chapter 9 describes for the original contribution.

The throughline is the same one from the Introduction's ROI argument: GRC's total effort across a use case's life doesn't disappear, but its shape changes from one expensive, high-stakes review concentrated at the end to a series of smaller, well-defined touchpoints spread across a process that was designed, from the estimation stage onward, to produce exactly the artifacts each touchpoint needs.

Chapter 6 — AIGF and AIUC-1: Differences and Complementarity

None of this makes external certification irrelevant — it changes what that certification is built on. It's worth being precise about how AIGF-driven Governance-as-Code relates to AIUC-1, the agent security and reliability standard from the Artificial Intelligence Underwriting Company, because the two operate at different layers and answer different questions, and because confusing them is one of the more common mistakes an FSI organization makes when planning its governance program.

6.1 Two Different Questions, Two Different Layers

AIUC-1 is a point-in-time, audited, insured attestation. It combines an audit of organizational controls with adversarial testing of the product itself, backed by Lloyd's of London insurance — a structure that gives the certifier real financial exposure if a certified agent fails, refreshed quarterly to keep pace with how fast agents change. It's an external signal: proof to a regulator, a board, or a counterparty that a specific agent has been independently vetted.

AIGF, running through the Governance-as-Code pipeline, is the continuous, internal engine that produces that proof as a byproduct of how the system was built. It doesn't issue a certificate. It defines the risks and controls, enforces them automatically through the Controls stage's CI/CD checks and CCC, orchestrates agentic workflows through Fluxnova, and observes the running system in production.

Put simply:

- **AIGF answers "how do we build and run this responsibly, every day, for every agent?"** — it's the pipeline.
- **AIUC-1 answers "can we prove this specific agent is safe to someone outside the institution?"** — it's the credential.

6.2 A Closer Look at AIGF

AIGF is not a product an FSI organization buys or a body that issues verdicts — it's a shared, open-source artifact that an FSI organization's own Governance Layer uses to run its own program. Three things define what it actually is:

- **It's a catalog, not a checklist someone hands you.** The risk catalog and use-case taxonomy (Chapters 1 and 9) are structured, versioned content in a public GitHub repository. A FSI organization's Gover-

nance Layer reads it, references specific risk IDs in its own CALM controls (Chapter 10), and can propose additions back to it.

- **It's governed by consensus, not by a vendor.** Decisions about what goes into the catalog are made by a FINOS working group — the same FSI organizations and technology providers that use the framework, including Morgan Stanley, Citi, and NatWest among others. There's no company selling AIGF compliance; there's a community maintaining a shared reference.
- **It produces no certificate.** Running a use case through AIGF-driven Governance-as-Code doesn't result in a document an FSI organization can hand a counterparty saying "AIGF-certified." What it produces is the CI/CD history, the CALM specification, the execution traces, and the risk mapping — the raw evidence Chapter 10 and Chapter 11 built out for the KYC use case — which is exactly the evidence an AIUC-1 audit, or an internal audit, would otherwise have to go and collect by hand.

6.3 A Closer Look at AIUC-1

AIUC-1 is a specific, named certification standard with its own defined structure — worth understanding on its own terms rather than only in contrast to AIGF:

- **It's organized around six pillars.** Public descriptions of the standard group its requirements under security, safety, reliability, accountability, data and privacy, and societal risk — a broader frame than a typical security-only compliance standard.
- **Certification means passing an audit and adversarial testing, not filling out a questionnaire.** Vendors pursuing AIUC-1 undergo independent, accredited audits combined with adversarial red-teaming aimed at surfacing failure modes — unauthorized tool calls, jailbreaks, and similar attacks — before a certificate is issued.
- **The insurance is the distinguishing feature.** Because AIUC (the company) both certifies and underwrites the risk of certified agents through Lloyd's of London, a failure in a certified agent creates direct financial exposure for the certifier — a structurally different incentive than a standards body that only issues a pass/fail opinion.
- **It refreshes quarterly, not annually.** Given how quickly agent behavior and the surrounding threat landscape shift, a certification that was accurate a year ago says very little about an agent today — so AIUC-1 re-tests on a shorter cycle than most legacy compliance certifications.

6.4 Side by Side

Aspect	AIGF / Governance-as-Code	AIUC-1
What it is	An open-source risk catalog, use-case taxonomy, and governance methodology, enforced through a pipeline.	An audited, insured certification standard for individual AI agents.
Governed by	A FINOS open working group — FSI and vendor participants, consensus-based.	The Artificial Intelligence Underwriting Company (AIUC), a private firm.
Basis of trust	Transparency — anyone can read the risk catalog, the taxonomy, and the controls.	Independent audit plus adversarial testing, backed by Lloyd's of London insurance.
Cost	Free and open source.	A paid certification and audit engagement.
Output	Continuous evidence, generated automatically as a byproduct of the pipeline.	A point-in-time certificate, refreshed quarterly.
Scope	Every AI and agentic use case across the institution.	Specific agents chosen for certification, one at a time.
Applies to	Build and buy — any use case, built internally or sourced from a vendor.	Any AI agent, though typically pursued for customer-facing or high-autonomy ones.
Recognized by	Peer financial institutions and regulators familiar with FINOS's mappings.	Boards, customers, and counterparties wanting independent, insured proof.

6.5 Using Them Together

An FSI organization that has already embedded AIGF into its Governance-as-Code pipeline isn't starting from zero when it decides a given agent needs AIUC-1 certification. The reference architecture already maps to known threat models. The CI/CD history already shows policy validation over time. The observability layer already has the telemetry an auditor would otherwise have to request manually. Rather than treating certification as a separate, bolt-on project, it becomes a natural checkpoint layered on top of governance that was already running — which is precisely the relationship the FINOS AI Fund gestures at when it lists certification and market-readiness support as a complement to, not a replacement for, governance-as-code enablement.

6.6 Frequently Asked Questions

- **Do we need both AIGF and AIUC-1?** For most FSI organizations, yes, but not for every use case. AIGF-driven Governance-as-Code is the baseline every agentic use case should run through — it's free, and Chapter 7's task lists assume it's already in place. AIUC-1 is a targeted, paid step reserved for specific agents — Chapter 7's Step 9 task list frames this as a deliberate decision, not a default.
- **If we already run AIGF, do we still need AIUC-1 for a given agent?** Depends on who needs convincing. AIGF's artifacts satisfy an internal audit and demonstrate a mature program. But AIGF alone gives no third-party attestation — no independent auditor has verified anything, and no insurer has staked money on the outcome. If a board, a major customer, or a regulator wants proof that doesn't rest on the FSI organization's own word, that's what AIUC-1 is for.
- **Can AIUC-1 replace our internal risk assessment?** No. AIUC-1 certifies a specific agent at a point in time; it doesn't classify use cases, maintain a risk catalog, or tell an engineering team which controls to build in. An FSI organization without an internal program like AIGF's would have nothing to show an AIUC-1 auditor except the agent itself — no risk mapping, no CALM specification, no CI/CD history — which is exactly the gap Chapters 9 and 10 exist to fill before Chapter 11 even starts building.
- **Does AIUC-1 only apply to vendor-bought agents?** No — it applies to any AI agent, whether built in-house or bought from a vendor. A FSI organization's own KYC agent from Chapters 9 through 11 is just as eligible for AIUC-1 certification as a third-party chatbot would be, provided it clears the same audit and testing.
- **Who's actually accountable if something goes wrong — FINOS or AIUC?** Neither takes on the FSI organization's regulatory accountability in either case; an FSI organization remains responsible for its own agents regardless of which of these it uses. The difference is in what each party puts at risk alongside that: FINOS, as an open-source foundation, doesn't issue certificates or take on financial exposure. AIUC does — the insurance backing described in Section 6.3 is AIUC's own money at stake if a certified agent fails, which is not something running AIGF alone gives an FSI organization access to.
- **What happens if we pass AIUC-1 but our AIGF risk catalog is out of date?** The certificate reflects a point in time — Section 6.3's quarterly refresh exists precisely because a stale certificate is a real risk. An out-of-date internal risk catalog is a governance gap regardless of certification status, and it's the kind of gap Chapter 5's continuous-monitoring argument (Section 5.2) is built to catch before a quarterly

re-audit would.

- **Is AIUC-1 required by any regulation?** No — it's an industry-developed standard, not a legal requirement. Its value is in being independently verifiable and insured, which can make regulatory conversations easier, but it doesn't substitute for whatever an FSI organization's actual regulatory obligations are under frameworks like the EU AI Act or existing model-risk-management guidance.
- **Does running AIGF give us any external proof at all, without AIUC-1?** Indirectly, yes. Because AIGF's risk catalog is explicitly mapped to recognized frameworks (Chapter 5), an FSI organization can show a regulator that its internal program aligns with those frameworks. What it can't offer is an independent auditor's signature or an insurer's financial backing — that's specifically what Section 6.1 means by AIGF being the pipeline and AIUC-1 being the credential.

Chapter 7 — A Step-by-Step Path Through the Pipeline

All of this can sound abstract until you walk through it as a sequence. Here's how an FSI organization actually moves a new agentic use case from idea to governed, running system, starting where AIGF itself starts: classification.

7.1 Teams and Roles

Chapters 4 and 5 used simple shorthand — "Engineering," "GRC" — to keep the focus on what changes about the work itself. In a real FSI organization, that shorthand maps onto a specific and fairly consistent organizational pattern, and it's worth being precise about it before walking through who does what at each step.

FSI organizations rarely organize AI work into a single department. Because financial services is regulated and deeply specialized, most institutions use a hub-and-spoke model that splits AI talent across three groups:

- **The Hub — a central AI Center of Excellence.** A single technology unit, usually reporting to a Chief Data and Analytics Officer or Head of AI, that serves the whole institution without building any one business line's product. In this guide's pipeline, the Hub is who owns the reference architecture library and CALM patterns (Section 4.1), and who builds and operates the CI/CD integration, Common Cloud Controls, and the observability layer (Chapters 1 and 2) — standardized once, centrally, and reused by every business line rather than rebuilt for each use case.
- **The Spokes — embedded AI squads inside each business line.** Cross-functional teams sitting inside retail and digital banking, risk and corporate lending, financial crime and fraud, or treasury and operations, each with the domain knowledge to build the actual use case. A Spoke's product manager and domain expert own the business justification and the go-live decision; a Spoke's AI engineers write the CALM specification and the Fluxnova process definition for their own use case, building on what the Hub provides rather than standing up their own infrastructure. When Chapters 9 and 10 develop a document-review use case, that work sits inside a Spoke — plausibly the financial crime and fraud team, given the use case.
- **The Governance Layer — sitting horizontally across both.** This is worth splitting into two distinct functions rather than treating as one "GRC" block, because they check different things: **Model Risk Management (MRM)** is an independent second-line function that vali-

dates and stress-tests a model's technical behavior before it reaches production — the audience for the heuristic risk assessment in Step 2 and the DMN table review in Step 5. **Legal, Compliance & Ethics** checks regulatory alignment, bias, and data-privacy entitlements — the audience for the taxonomy classification in Step 1 and the certification decision in Step 9. A use case can clear MRM's technical validation and still need a Compliance sign-off on a point neither team's charter covers alone.

Four labels carry through the rest of this chapter: **Hub — Architecture**, **Hub — MLOps / DevSecOps**, **Business Spoke — AI Engineering** (paired with a Spoke's Product/Domain owner where the task is a business decision rather than a technical one), and **Governance Layer**, split into MRM and Compliance where the distinction matters. There's no standalone QA label — testing the pipeline itself falls to Hub MLOps, while testing the specific use case's behavior falls to the Spoke's AI engineers, which is where that work actually happens in this structure.

This structure is also exactly why Governance-as-Code matters in the way Chapter 1 argued. A hub-and-spoke model run without it tends to produce precisely the pitfalls the Introduction opened with: Spokes moving fast but reinventing risk assessment independently, or the Hub becoming a bottleneck reviewing every Spoke's bespoke approach one at a time. With AIGF's taxonomy, risk catalog, and reference architectures standardized once at the Hub and the Governance Layer, every Spoke reuses the same artifacts instead of negotiating its own — which is what lets the FSI organization, in the framing from the Introduction, innovate quickly at the edge while staying compliant at the core.

7.2 An Illustrative Timeline

The chart below shows one realistic timeline for a first-time use case moving through all nine steps, with the team primarily driving each one. It's illustrative, not a commitment — a simpler use case compresses this considerably, and Chapter 9's point about reuse holds here too: an FSI organization's second and third use cases move faster than this because the taxonomy, risk catalog, and reference architectures this timeline assumes are being built from scratch are already sitting there the second time around.

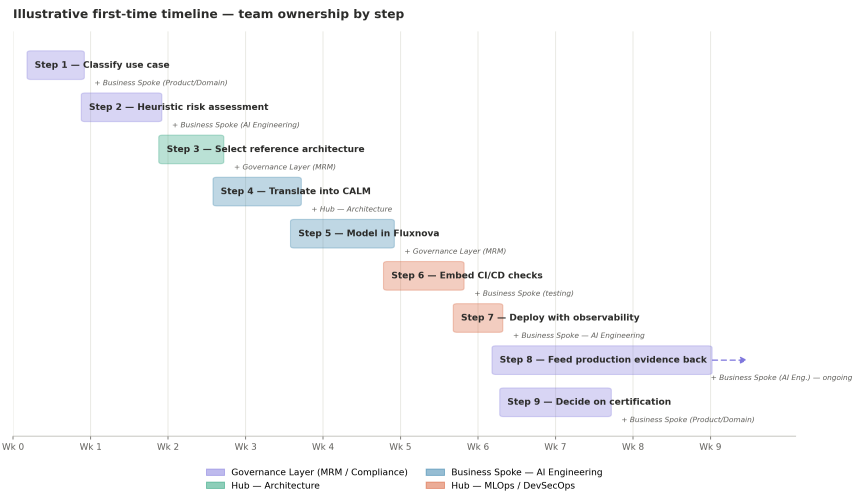


Figure 3. An illustrative first-time timeline. Bars show a step's primary drive time; italic labels note the main supporting team. Steps 8 and 9 begin once the system is live and run in parallel with each other.

A few things the chart is deliberately showing. Steps overlap rather than running strictly in sequence — Step 3's architecture selection is still being finalized as Step 4's CALM authoring begins, because in practice a team drafts the specification against a near-final architecture rather than waiting for a formal sign-off email. Step 8 (production feedback) is drawn as ongoing, with no end date, because it isn't a task that finishes — it's the steady state the system lives in from launch onward, which is precisely the point made in Chapter 1's feedback loop and Chapter 5's discussion of production monitoring. Step 9 (the certification decision) overlaps with the start of Step 8 rather than waiting for it, since the evidence needed for that decision starts accumulating the moment the system goes live.

7.3 Iterative Delivery, Not a Cascade

Figure 3 draws Steps 1 through 6 as a single relay for the whole use case: a full risk assessment, then a full architecture, then a full CALM specification, and so on, each waiting on the last. That's a fair way to explain what each step produces, but taken literally as a project plan it's a heavy commitment — anything the team learns late means walking back up the chain. Steps 7 through 9 already avoid this: Step 8 has no end date, and Step 9 overlaps with Step 8's start rather than waiting for it. The same thinking extends naturally to Steps 1 through 6, and it's worth spelling out how.

The shift is to slice the use case by behavior rather than by phase. Instead of fully specifying the entire use case before writing any CALM or Fluxnova code, a team takes a thin behavioral slice — the happy path first, then the

escalation path, then accumulated edge cases — and runs all six steps, in miniature, for that slice alone. Each pass is a small, complete loop: classify the slice, assess its risk, confirm the architecture still fits, extend the CALM specification, extend the Fluxnova process, and let CI/CD validate it — all before moving to the next slice.

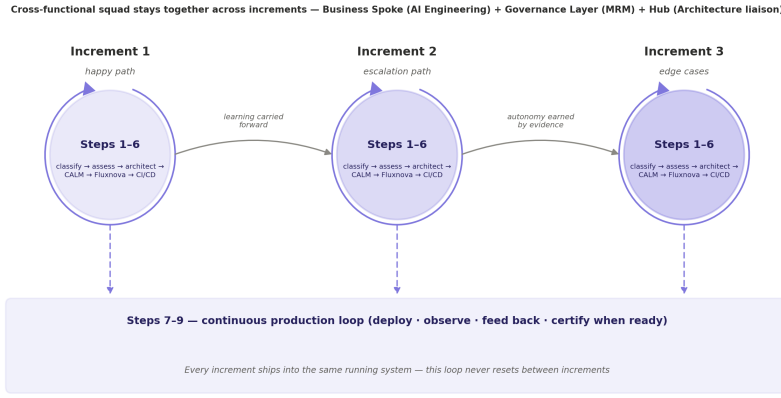


Figure 4. Iterative delivery: Steps 1–6 run as a small loop per behavioral increment, with shading indicating how much autonomy each increment has earned. Every increment ships into the same continuous Steps 7–9 production loop from Figure 3 — it never resets.

Three things make this work in practice, each visible in the diagram:

- **The same squad runs every loop.** A cascade model matches a relay-race org chart — Hub Architecture finishes and hands off to the Spoke, which hands off to Hub MLOps, which hands off to Governance at the end. Iterative delivery instead wants a standing cross-functional squad — a Business Spoke AI engineer, a Governance Layer (MRM) reviewer, and a Hub Architecture liaison — sitting together for the duration and cycling through Steps 1–6 repeatedly, so Governance participates continuously rather than gating at the end. This is the practical form of the argument Section 5.2 already made about GRC's involvement spreading across the lifecycle.
- **Autonomy is earned by evidence, not assumed upfront.** Rather than fully specifying every DMN rule in Step 5 before anything ships, the first increment's decision table can start maximally conservative — effectively "escalate everything" — and later increments relax specific rules toward auto-clear only once Step 8's production evidence justifies it. The risk model itself gets built iteratively, which is arguably safer than designing it fully upfront, since autonomy only ever gets added where there's evidence to support it.

- **Steps 1 through 3 get fast with repetition.** Chapter 9's argument about the taxonomy and risk catalog being reusable infrastructure applies within a single use case's increments, not just across different use cases: the second and third increments' classification and architecture checks are quick confirmations that nothing has changed, not full workshops repeated from scratch.

Two things don't get to loosen just because delivery is iterative. Every increment still needs its classification and risk checkpoint, however brief — skipping Step 1's spirit to move faster is exactly the kind of shortcut the Introduction's pitfalls warned about. And every CALM or DMN change, no matter how small the increment, goes through the same version-controlled review as a full specification would; an informal, un-reviewed increment quietly erodes the audit trail Chapter 11 depends on. Certification (Step 9) also stays a genuine gate near the point a use case is considered ready — a system can't be partially certified — but because evidence has been accumulating continuously across increments rather than in one final phase, that decision is cheaper to make whenever the team reaches it.

Step 1 — Classify the use case with the AIGF Use Cases Taxonomy.

Before anything is built, the use case gets placed against AIGF's taxonomy, which categorizes AI systems by type, architecture pattern, autonomy level, tool authority, and data-handling needs. An internal coding assistant with no customer data access sits in a very different risk band than an autonomous agent that can initiate payments or advise clients directly. Context shapes risk tolerance directly: an AI helping an internal developer can tolerate more error than one advising clients on investments.

- **Write the one-paragraph use case narrative** — Business Spoke (Product/Domain)
- **Run the taxonomy classification workshop** — Governance Layer — Compliance, with the Business Spoke and Hub Architecture present
- **Record the classification as a reviewable artifact** — Governance Layer — Compliance

Step 2 — Run the heuristic assessment against the risk catalog.

With the use case classified, the team works through AIGF's eight-step heuristic assessment methodology, which walks systematically through what actually drives risk for that specific system: what data the system is trained on, what data flows in through prompts and queries, and what

could plausibly come out. The output is a shortlist of which risks genuinely apply and which mitigations from the catalog address them.

- **Walk the eight-step heuristic methodology against the use case** — Governance Layer — MRM
- **Identify which catalog risks genuinely apply** — Governance Layer — MRM, with Business Spoke AI Engineering input on technical feasibility
- **Draft the mitigation mapping** — Governance Layer — MRM and Business Spoke AI Engineering jointly

Step 3 — Select the matching reference architecture and threat model.

The taxonomy classification and risk assessment feed directly into FINOS's AI Reference Architecture Library. Because threat models vary by autonomy level, tool authority, and human-oversight pattern, the classification from Step 1 determines which pre-vetted architecture pattern the team should build from.

- **Match the taxonomy classification to a reference architecture pattern** — Hub — Architecture
- **Review the pattern's threat model against Step 2's risk shortlist** — Hub — Architecture and Governance Layer — MRM
- **Sign off on the architecture selection** — Hub — Architecture lead

Step 4 — Translate the architecture into CALM specifications.

The chosen reference architecture gets expressed through CALM as a machine-readable, executable specification — the bridge between the regulatory-and-risk language of Steps 1–2 and the actual code the engineering team will write.

- **Author the CALM nodes and relationships (Section 4.1)** — Business Spoke — AI Engineering, with Hub — Architecture
- **Attach controls mapped to Step 2's mitigations** — Business Spoke — AI Engineering, reviewed by Governance Layer — MRM
- **Merge the CALM file through a standard pull-request review** — Business Spoke — AI Engineering

Step 5 — Draw the deterministic boundaries and hand them to Fluxnova.

Using the risk assessment and architecture as a guide, the team identifies which parts of the workflow must be deterministic — mandatory approval gates, escalation thresholds, sanctions or compliance checks — and which parts can safely be left to the LLM's judgment. Those deterministic boundaries get modeled as BPMN process flows and DMN decision tables in Fluxnova.

- **Model the BPMN process — tasks, gateways, end states** — Business Spoke — AI Engineering
- **Author the DMN decision tables against signed-off outcomes** — Business Spoke — AI Engineering and Governance Layer (MRM and Compliance)
- **Configure the agentic ad-hoc subprocess (prompt, tools, restricted variables)** — Business Spoke — AI Engineering
- **Review the deterministic boundaries against Step 2's mitigations** — Governance Layer — MRM

Step 6 — Embed compliance checks into CI/CD.

As engineering builds against the CALM specification, CALM-driven compliance checks run automatically in the CI/CD pipeline, continuously validating the emerging system against the AIGF-defined policies selected back in Step 2. Common Cloud Controls does the equivalent for the underlying cloud and infrastructure layer.

- **Wire CALM and Fluxnova validation into the build pipeline** — Hub — MLOps / DevSecOps
- **Define release gates that block on a control failure** — Hub — MLOps / DevSecOps, with Governance Layer — MRM
- **Test the pipeline against known-good and known-bad cases** — Hub — MLOps / DevSecOps, with Business Spoke AI Engineering for domain-specific cases

Step 7 — Deploy with observability switched on from day one.

Once running, the system is monitored through the pipeline's observability layer — OpenTelemetry instrumentation feeding dashboards (Grafana being a common choice) — capturing real-time behavior, cost, and Fluxnova's execution traces of every workflow instance.

- **Instrument the system with OpenTelemetry** — Business Spoke — AI Engineering and Hub — MLOps / DevSecOps
- **Stand up dashboards for operational and compliance views** — Hub — MLOps / DevSecOps
- **Define evals and the drift thresholds that trigger a review** — Governance Layer — MRM and Business Spoke AI Engineering
- **Approve go-live** — Business Spoke (Product/Domain) and Governance Layer — Compliance

Step 8 — Feed production evidence back into governance.

The observability data, execution traces, and any evals run in production feed back into the use case's risk profile, closing the loop. If the agent's behavior in production surfaces a risk the original assessment didn't anticipate, that goes back into Step 2 for reassessment — and, at the framework level, back into AIGF's own catalog through FINOS's open, collaborative maintenance process.

- **Monitor dashboards and evals on an ongoing basis** — Governance Layer — MRM, with Business Spoke AI Engineering
- **Triage any finding that falls outside expected behavior** — Governance Layer — MRM and Business Spoke AI Engineering
- **Update the risk mapping or DMN tables where warranted** — Governance Layer — MRM and Business Spoke AI Engineering
- **Consider contributing the finding back to AIGF (Chapter 9)** — Governance Layer — Compliance, with the Hub representing the FSI organization's wider interest

Step 9 — Decide whether external certification is warranted.

With all of the above in place, the FSI organization is now in a strong position to make a deliberate call: does this specific use case warrant pursuing AIUC-1 certification on top of everything already governed internally? Because the reference architecture, CI/CD validation history, and execution traces already exist, this becomes a targeted decision layered on mature governance, not a scramble to produce evidence that was never being captured.

- **Assess whether the use case's risk profile warrants AIUC-1** — Governance Layer — Compliance and Business Spoke (Product/Domain)

- **Assemble the evidence package (CI/CD history, execution traces)** — Governance Layer — MRM and Hub — MLOps / DevSecOps
- **Engage an accredited auditor if pursuing certification** — Governance Layer — Compliance

The throughline across all nine steps is that nothing here is a one-off exercise. The taxonomy, the risk catalog, the architecture library, and the CI/CD checks all persist as reusable infrastructure — so the second use case an FSI organization runs through this pipeline moves faster than the first, and the fiftieth moves faster still.

Chapter 8 — Getting Teams Trained and Ready

Chapter 7 described who does what once a pipeline is running and how to sequence the work. This chapter addresses the question that comes before that: how does a Chief AI Officer get an organization that doesn't yet think this way to the point where Chapter 7 is executable at all. Training and rollout is a different problem from process design, and treating them as the same problem is itself one of the pitfalls from the Introduction — a Spoke "moving fast" in an ungoverned way usually just means nobody trained it on the taxonomy yet.

8.1 Training Is Role-Specific, Not a Single Curriculum

The six roles from Section 7.1 don't need the same thing from a training program, and treating them as if they do wastes effort on the roles that need it least while underserving the ones that need it most.

- **Governance Layer** (**MRM and Compliance**) needs the deepest investment, and it's the one group where structured, scheduled sessions are worth running: working through the risk catalog and taxonomy repeatedly until classifying a new use case takes an afternoon, not a workshop. This group's fluency is what determines whether Steps 1 and 2 are fast or slow for every use case that follows it, which makes it the highest-leverage place to invest first.
- **Business Spoke AI engineers** learn CALM and Fluxnova by writing them under review, not by attending a course. The most effective "training" here is the first pilot use case itself, done deliberately slowly with an experienced reviewer pairing on every pull request.
- **Hub Architecture and Hub MLOps** need the least ramp-up if they're already doing platform engineering work — the shift for them is conceptual more than technical: governance now lives in a CALM file and a CI/CD check, not in a wiki page nobody reads.
- **Business Spoke product managers and domain experts** need the lightest touch: enough literacy to write a use case narrative and understand why classification happens before any building starts, so Step 1 reads as useful rather than as bureaucratic friction.

8.2 Formal Training: Linux Foundation Education and Service Providers

Section 8.1 assumed an FSI organization builds its entire training path from scratch. It doesn't have to. Linux Foundation Education — the Linux Foundation's training and certification arm, which already runs FINOS's existing certification programs — offers formal courses and

certifications covering FINOS projects, with coverage of AIGF, CALM, and Fluxnova alongside them as those projects mature. Beyond the self-paced catalog, instructor-led training is also available through Linux Foundation training partners and FINOS member service providers, and FINOS itself has begun running hands-on leader training aimed specifically at risk, compliance, legal, model governance, and operations leaders operationalizing this pipeline. For a CAIO standing up a program from nothing, this is worth treating as a foundational layer underneath Section 8.1's role-specific approach, not a substitute for it:

- **Formal courses build shared vocabulary fast.** A Governance Layer analyst who has already worked through a structured course arrives at the first taxonomy classification workshop fluent in the categories, rather than learning the taxonomy's shape for the first time under deadline pressure.
- **They fit naturally with the roles Section 8.1 flagged as needing the lightest touch.** Business Spoke product managers and domain experts, who only need enough literacy to write a use case narrative and understand why classification matters, are well served by a short formal course rather than a bespoke internal session the Governance Layer has to build and maintain itself.
- **Service providers fill the gap a self-paced course can't.** Hands-on, instructor-led sessions tailored to an FSI organization's own environment are useful when a Hub team is standing up CALM and Fluxnova tooling for the first time and wants an expert in the room rather than documentation alone — and FINOS's own leader training exists precisely because Governance Layer roles benefit from guided, cohort-based sessions more than a self-paced course.
- **None of this replaces Section 8.3's argument.** A completed course demonstrates familiarity with the concepts; writing a CALM specification that clears review demonstrates the capability. Formal training gets a squad to its first pilot faster and with fewer false starts — it is not, on its own, a finish line.

8.3 The First Use Case Is the Training Program

The natural instinct is to train everyone first, then start a pilot. It's worth inverting that. Pick one real, modest use case, assemble the standing cross-functional squad described in Section 7.3, and let that squad's first pass through Steps 1 through 6 be the training — documented deliberately as it happens, specifically so it becomes the FSI organization's own internal playbook layered on top of AIGF's generic guidance. This is Chapter 9's argument about the first use case becoming a template, applied to people instead of artifacts: the KYC use case in Chapters 9 through 11 is exactly the

kind of pilot this section has in mind, and an FSI organization's own first pass through it produces the institution-specific version of that template that every subsequent Spoke actually learns from.

This also sets a realistic maturity progression rather than a single rollout event: one pilot squad completes a use case and writes down what it learned; that documentation, plus the reusable taxonomy and architecture artifacts, lets a second Spoke run its own use case with lighter support; by the third or fourth use case, the pattern is established enough that a new Spoke can largely self-serve from what previous ones produced, with the Governance Layer reviewing rather than teaching from scratch each time.

8.4 Get a Seat in the Room Early

AIGF is a living, community-governed project, not a specification that gets downloaded once and left alone — it has already gone through a major version change that added agentic-specific risk coverage. A CAIO who sends someone to FINOS's regular working group meetings (Chapter 9 mentions these as part of the contribution process) gets two things at once: the FSI organization's internal understanding stays current as the framework evolves, and the FSI organization gets a voice in what the catalog covers next, rather than finding out about a change after the fact. This is a natural complement to Section 8.2's formal training paths — a course teaches the framework as it stood when the course was recorded; a seat in the working group is how the FSI organization's understanding keeps pace with a framework that keeps moving. Treating AIGF adoption as a one-time download, formal course included, is a slower-motion version of the staleness problem Chapter 6's FAQ raises about certifications going stale between audits — the fix is the same in both cases: stay continuously engaged rather than checking in periodically.

8.5 The Failure Mode Specific to Training: Skew

Beyond the general pitfalls in the Introduction, rollout has one failure mode worth naming on its own: investing heavily in engineering enablement — because it's tangible, has a curriculum, and feels like real training — while leaving the Governance Layer and Business Spokes undertrained. This doesn't remove the bottleneck an FSI organization is trying to fix; it just moves it. An engineering team that can write a CALM specification quickly gains nothing if the Governance Layer still needs three weeks to classify the use case that CALM file depends on. A training investment should be sized to whichever role is currently the slowest step in Chapter 7's timeline, not to whichever role is easiest to run a workshop for — and the same imbalance can now happen inside Section 8.2's formal-training budget alone, if every seat goes to engineering courses and none to the Governance Layer

sessions FINOS runs specifically for that audience.

A related version of the same mistake is training once and considering the job done. Because AIGF itself changes (Section 8.4) and because an FSI organization's own playbook keeps accumulating lessons from each new use case (Section 8.3), training is closer to an ongoing rhythm than a single onboarding event — the same continuous-engagement principle Chapter 5 argues for GRC's involvement across a use case's lifecycle applies equally to how an organization stays current on the framework itself.

Chapter 9 — Codifying a Use Case

Step 1 deserves more attention than a single paragraph, for a specific reason: it's not just the first step, it's the *template* for how every subsequent use case gets handled. Get the pattern right once, and every team after you is filling in a known shape rather than inventing a process from scratch. This chapter walks through that pattern in full, using a KYC use case as the worked example, all the way through to opening a pull request against the AIGF repository itself.

9.1 Why This Step Is the Template

Everything downstream in the pipeline — the reference architecture in Step 3 (Chapter 10), the CALM specification in Step 4 (also Chapter 10), the Fluxnova process boundaries in Step 5 (Chapter 11), even the decision on AIUC-1 in Step 9 — takes its inputs from what gets produced in Step 1. If the use case is classified loosely or the risk assessment is shallow, every later stage inherits that weakness. So the goal of Step 1 isn't just to produce a classification — it's to produce a small set of concrete, reusable artifacts that the rest of the pipeline can consume mechanically, and that the next team building a similar use case can copy as a starting point.

Four artifacts come out of a well-run Step 1:

1. A **use case narrative** — what the system does, who it serves, and what decisions or actions it's permitted to take.

1. A **taxonomy classification** — where the use case sits against AIGF's Use Cases Taxonomy: type, architecture pattern, autonomy level, tool authority, human-oversight pattern, and data-handling requirements.

2. A **risk shortlist** — the specific entries from AIGF's risk catalog that actually apply, produced through the eight-step heuristic assessment.

3. A **mitigation mapping** — the catalog mitigations selected against each shortlisted risk, with enough detail that an engineer can act on them directly.

9.2 Worked Example: An Agentic KYC Document Review Assistant

Take a concrete case: an FSI organization wants to build an agent that reviews KYC onboarding documents (passports, proof-of-address, corporate registry extracts), extracts and validates the required fields, cross-checks them against sanctions and PEP lists, flags discrepancies, and either auto-clears low-risk cases or escalates ambiguous ones to a human analyst.

Narrative

In a sentence: an agent that triages KYC documentation for new retail and corporate customers, with autonomy to clear straightforward cases and a hard requirement to escalate anything ambiguous, high-risk, or sanctions-adjacent to a human analyst.

Taxonomy classification

- *Type* — a document-processing and decisioning agent, not a conversational assistant.
- *Architecture pattern* — single agent with tool access (OCR/extraction tool, sanctions-list lookup, case management system), not a multi-agent system.
- *Autonomy level* — bounded autonomy: the agent can approve low-risk cases outright but cannot reject or escalate-close a case without a human decision, and cannot override a sanctions hit under any circumstance.
- *Tool authority* — read access to document stores and sanctions/PEP databases, write access limited to a case management queue (it can move a case, not close a compliance file).
- *Human-oversight pattern* — human-in-the-loop for all escalations, human-on-the-loop (spot-check sampling) for auto-cleared cases.
- *Data-handling requirements* — processes PII and government ID data, so falls into the taxonomy's higher data-sensitivity band, with corresponding requirements for data minimization and retention limits.

This classification alone already tells you this is not a low-stakes internal tool — it's a customer-onboarding system touching regulated data with a real compliance function, which shapes everything from here on.

Heuristic assessment

- *Training and grounding data* — the agent isn't fine-tuned on customer data; it's a general-purpose model grounded via retrieval against the FSI organization's KYC policy documents and live sanctions feeds. That narrows one category of risk (training-data leakage) while keeping others open (retrieval-source poisoning, stale sanctions data).
- *Input data* — each query includes an uploaded document plus structured case metadata; both may contain full PII (name, DOB, national ID number, address).

- *Output and exposure risk* — the agent's outputs are structured decisions and extracted fields written into the case management system, not free text shown to the customer.
- *Context and risk tolerance* — this is a client-facing, regulatory-relevant use case with essentially zero tolerance for a false clear on a genuine sanctions match, and low tolerance for inconsistent treatment of similar cases.

Risk shortlist

Reviewing the risk catalog against that profile, a KYC use case will typically pull in risks across all three of AIGF's categories — operational, security, and regulatory & compliance. Concretely: data quality and drift (extraction accuracy degrading as document formats change), hallucinated or fabricated field values presented with false confidence, sanctions-list staleness or lookup failure, inconsistent decisioning across similar cases (a fairness and audit concern), and prompt injection via a maliciously crafted document. Not every risk in the catalog applies — that's the point of running the assessment rather than applying the whole catalog by default.

Mitigation mapping

Each shortlisted risk gets matched to catalog mitigations and made concrete for this use case — for example: mandatory human confirmation before any sanctions-adjacent case is cleared (not just flagged); confidence thresholds below which the agent must escalate rather than guess; periodic sampling and drift monitoring on extraction accuracy against a labeled test set; and input sanitization plus a locked-down instruction boundary around anything read from an uploaded document, so text embedded in a fake "certificate" can't act as a prompt.

9.3 Turning the Assessment into a Contribution

At this point the FSI organization has a complete internal risk assessment — useful on its own. Codifying it as a contribution to AIGF means expressing it as a structured artifact using the repository's existing template, rather than a one-off internal memo.

The AIGF repository (finos/ai-governance-framework on GitHub) keeps its published content — the risk catalog, the mitigation catalog, and the use case taxonomy — as structured documents under its docs/ folder, with a dedicated templates-for-ri-md folder providing the markdown templates contributors are expected to follow for new entries, plus scripts and scripts_docs that build and validate the published site from those markdown sources. Practically, contributing the KYC use case means:

- Taking the relevant template from templates-for-ri-md and filling it in with the narrative, taxonomy classification, risk shortlist, and mitigation mapping worked out above.
- Following the project's existing conventions for structure and terminology, documented in CONVENTIONS.md, so the entry reads consistently with everything already published.
- Running the project's local build/validation scripts (per DEVELOPMENT.md) before submitting, so the new entry doesn't break the generated site.

9.4 Making the Pull Request

The mechanics follow FINOS's standard open-source contribution model:

4. **Check for or open an issue first.** FINOS's contribution norms across projects favor discussing a proposed addition in an issue before code lands — for a new use case, this is where you'd sanity-check the classification and risk shortlist with the maintainers and the wider community before investing in the full write-up.
5. **Fork the repository and create a branch** for the new use case entry.
6. **Add the markdown file** using the appropriate template, following CONVENTIONS.md.
7. **Sign every commit with a DCO** (Signed-off-by: Name , or git commit -s) — this is a hard requirement across FINOS projects and commits without it get flagged automatically.
8. **Open the pull request**, referencing the issue from step 1, and let the automated checks (docs build, formatting) run.
9. **Engage with review.** AIGF is governed as an open working group — becoming a recognized "Participant" is as simple as attending a meeting or contributing to an issue or PR — and substantive changes typically get discussed at the project's bi-weekly meetings alongside the asynchronous GitHub review.
10. **Iterate and merge.** Once the maintainers (drawn from the project's committer team) are satisfied, the entry merges and becomes part of the published framework — visible to every other institution using AIGF, not just the FSI organization that wrote it.

9.5 Why This Is Worth the Effort

Two things happen once the KYC use case is merged. Internally, it becomes the literal template the FSI organization's next similar use case — say, ongo-

ing transaction monitoring, or a credit-decisioning assistant — starts from: the taxonomy dimensions are already understood, the risk categories that matter for regulated customer-data use cases are already known, and the mitigation patterns are already proven. Downstream in the pipeline, this artifact is exactly what Chapter 10 needs to select the right reference architecture and translate it into a CALM specification, and what Chapter 11 needs to know which decision points must be pinned down in Fluxnova as deterministic gates rather than left to the model.

Externally, the entry strengthens the shared catalog every other FINOS member institution draws from — which is the entire point of AIGF being open source rather than a proprietary internal framework: the next FSI organization building a KYC agent doesn't start from zero, because this one didn't either.

Chapter 10 — Solution Architecture in Depth: From Reference Pattern to CALM Specification

Chapter 9 produced four artifacts for the KYC use case: a narrative, a taxonomy classification, a risk shortlist, and a mitigation mapping. This chapter picks up exactly where that leaves off — Steps 3 and 4 — turning that classification into an actual architecture decision, then expressing that decision as a CALM specification concrete enough for engineering to build against and for Fluxnova (Chapter 11) to enforce.

10.1 Selecting the Reference Architecture (Step 3)

Chapter 9's taxonomy classification already did most of the work here: a single agent with tool access (an extraction tool and a sanctions/PEP lookup), bounded autonomy, and a mandatory human-in-the-loop touchpoint for escalations. That maps directly onto the AI Reference Architecture Library's single-agent, tool-using pattern — not the multi-agent pattern, and not a plain retrieval-augmented pattern, because there's exactly one reasoning agent involved and its job is bounded tool use rather than open-ended document synthesis.

Selecting this pattern is not a formality — it commits the team to a known threat model that comes with the pattern, before a single line of the use case's own code is written. The single-agent, tool-using pattern's threat model already flags tool-scope creep (the agent reaching for a tool it wasn't meant to have), prompt injection via content returned from a tool call, and human-review bypass as the risks any implementation of this pattern needs to address. None of that is specific to KYC — it's inherited the moment the pattern is chosen, which is the entire point of maintaining a reference architecture library rather than threat-modeling every use case from a blank page.

It's worth noting what would have happened with a different classification. Had Chapter 9 classified this as a multi-agent use case — say, a separate agent for extraction and a separate one for sanctions screening, coordinated by a third — a different pattern would apply, with a different threat model: inter-agent trust boundaries and coordinator compromise, rather than the single-agent pattern's tool-scope concerns. The architecture follows the classification; it is never chosen first and used to justify the classification after the fact.

Sign-off at this stage sits with Hub Architecture, checking one thing specifically: does the chosen pattern's default threat model actually cover everything in Chapter 9's risk shortlist, or does the use case have a risk the pattern doesn't anticipate. Here, it does: the pattern's standard threat model has nothing to say about a sanctions match specifically, because that's a

KYC-specific regulatory concern, not a generic tool-using-agent concern. That gap doesn't block the architecture selection — it becomes a control the team adds explicitly in Section 10.3, rather than something the pattern was ever going to hand over for free.

10.2 What the Pattern Leaves for This Use Case to Decide

This is the concrete version of the point made in Section 4.2: a reference architecture pattern, like a design pattern, gives capability structure without settling compliance specifics. Choosing the single-agent, tool-using pattern says the agent will reason over a bounded set of tools with a human checkpoint somewhere in the loop — it does not say which tools, what the checkpoint actually blocks, or which of Chapter 9's mitigations become enforceable rules rather than aspirations. Those decisions are what the CALM specification in Section 10.3 exists to pin down: exactly two tools (extraction, sanctions lookup), exactly one downstream system the agent can write to (the case management system), and exactly which of Chapter 9's mitigations get expressed as CALM controls rather than left as prose in a risk assessment document.

10.3 Translating the Pattern into CALM (Step 4)

With the pattern selected and its gaps identified, the architecture gets expressed as an actual CALM specification — the same nodes/relationships/controls structure introduced generically in Section 4.1, now applied for real. It's tempting to stop at the business-logic nodes — the agent, its two tools, the system it writes to — but a specification that only shows those leaves out exactly what a reviewer or an auditor asks about next: what actually executes this, and whose model is doing the reasoning. A complete architecture names the infrastructure too: the Fluxnova engine and its DMN decision service that Chapter 11 builds out in full, and the LLM provider platform the subprocess calls for reasoning.

```
{
  "nodes": [
    {
      "unique-id": "kyc-agentic-subprocess",
      "node-type": "service",
      "name": "KYC document review (agentic ad-hoc subprocess)",
      "description": "LLM-driven subprocess bounded to two inner tools: field
  extraction and sanctions/PEP lookup",
      "interfaces": []
    },
    {
      "unique-id": "kyc-extraction-tool",
```

```
    "node-type": "service",
    "name": "KYC field extraction tool",
    "description": "Extracts structured fields from an uploaded identity
document"
  },
  {
    "unique-id": "sanctions-lookup-tool",
    "node-type": "service",
    "name": "Sanctions / PEP lookup tool",
    "description": "Queries sanctions and PEP lists for a given identity"
  },
  {
    "unique-id": "kyc-case-management",
    "node-type": "system",
    "name": "Case management system",
    "description": "Receives the subprocess's structured output and the
process's routing decision"
  },
  {
    "unique-id": "fluxnova-engine",
    "node-type": "system",
    "name": "Fluxnova orchestration engine",
    "description": "Executes the KYC BPMN process definition and hosts the
agentic ad-hoc subprocess (Chapter 11)"
  },
  {
    "unique-id": "fluxnova-dmn-service",
    "node-type": "service",
    "name": "Fluxnova DMN decision service",
    "description": "Evaluates the risk & confidence decision table at the
process's first gateway (Section 11.5)"
  },
  {
    "unique-id": "llm-provider",
    "node-type": "service",
    "name": "LLM provider platform",
    "description": "Hosted large language model service the agentic
subprocess calls for reasoning; provider and model selected via Fluxnova's
agent configuration (Section 11.2)"
  }
],
"relationships": [
  {
    "unique-id": "agent-calls-extraction",
    "connects": { "source": "kyc-agentic-subprocess", "destination":
"kyc-extraction-tool" },
    "description": "Agent invokes extraction as a bounded tool call"
```

```
    },
    {
      "unique-id": "agent-calls-sanctions",
      "connects": { "source": "kyc-agentic-subprocess", "destination":
  "sanctions-lookup-tool" },
      "description": "Agent invokes sanctions/PEP lookup as a bounded tool
  call"
    },
    {
      "unique-id": "agent-writes-case",
      "connects": { "source": "kyc-agentic-subprocess", "destination":
  "kyc-case-management" },
      "description": "Structured output (fields, confidence, match status) is
  written to the case record"
    },
    {
      "unique-id": "engine-hosts-subprocess",
      "connects": { "source": "fluxnova-engine", "destination":
  "kyc-agentic-subprocess" },
      "description": "Fluxnova engine executes and hosts the agentic ad-hoc
  subprocess as part of the KYC BPMN process"
    },
    {
      "unique-id": "engine-uses-dmn",
      "connects": { "source": "fluxnova-engine", "destination":
  "fluxnova-dmn-service" },
      "description": "Engine invokes the DMN service to evaluate the risk &
  confidence gateway"
    },
    {
      "unique-id": "agent-calls-llm",
      "connects": { "source": "kyc-agentic-subprocess", "destination":
  "llm-provider" },
      "description": "Agent sends prompts to the configured LLM provider for
  reasoning inside the subprocess boundary"
    }
  ]
}
```

Notice what adding these three nodes changes. The business-logic view (agent, tools, case management) shows what the use case does; the infrastructure view (the engine, the DMN service, the LLM provider) shows what it runs on and whose model is in the loop — and both now live in the same reviewable file rather than one living in a design document and the other living nowhere until an auditor asks. The llm-provider node in particular is what gives vendor risk and model governance something con-

crete to review: which provider, which model, and — per Section 10.4 — whether that choice is still the one Governance signed off on.

This is where Chapter 9's mitigation mapping stops being prose and starts being enforceable: the sanctions-override gap identified in Section 10.1, and the mandatory-human-review mitigation from Chapter 9.2, both become controls attached directly to the agent node. The infrastructure nodes get their own, separate control — this is deliberately not the same compliance block as the agent's, because it's enforced by Common Cloud Controls (Chapter 2) rather than by CALM's own graph.

```
{
  "nodes": [
    {
      "unique-id": "kyc-agentic-subprocess",
      "node-type": "service",
      "name": "KYC document review (agentic ad-hoc subprocess)",
      "controls": {
        "compliance": {
          "description": "Controls derived from Chapter 9's risk shortlist
  and mitigation mapping",
          "requirements": [
            {
              "control-requirement-url": "https://fsi.example/controls/schema/
  no-sanctions-override.json",
              "control-config-url": "https://fsi.example/controls/config/kyc-n
  o-sanctions-override.json"
            },
            {
              "control-requirement-url": "https://fsi.example/controls/schema/
  mandatory-human-review.json",
              "control-config-url": "https://fsi.example/controls/config/kyc-c
  onfidence-threshold-95.json"
            }
          ]
        }
      }
    },
    {
      "unique-id": "fluxnova-engine",
      "node-type": "system",
      "name": "Fluxnova orchestration engine",
      "controls": {
        "infrastructure": {
          "description": "Common Cloud Controls mappings for the hosting
  environment (Chapter 2)",
          "requirements": [
            {
```

```
            "control-requirement-url": "https://fsi.example/controls/schema/
  ccc-workload-isolation.json",
            "control-config-url": "https://fsi.example/controls/config/kyc-f
  luxnova-hosting.json"
          }
        ]
      }
    }
  }
  ]
}
```

Nothing here is new mechanism — Section 4.1 already covered how a requirement and a configuration file combine. What's new is that both controls now trace directly back to a specific line in Chapter 9: "no override on a sanctions hit" is the architecture-level gap flagged in Section 10.1, and "mandatory human confirmation before any sanctions-adjacent case is cleared" is lifted verbatim from Chapter 9.2's mitigation mapping. A reviewer can trace every control in this file back to a specific, already-approved line in the risk assessment — there is no control here that wasn't first justified as a mitigation.

10.4 Architecture Review and Sign-off

Before engineering proceeds to Fluxnova modeling, this CALM file goes through a specific, narrow review — not a general architecture review, but a check against three things:

- **Coverage.** Does every mitigation in Chapter 9's mapping have a corresponding control in this file? A mitigation with no matching control is a mitigation that exists on paper only.
- **Scope match.** Does the relationships list match exactly the tool authority the taxonomy approved in Chapter 9 — no more, no less? An extra relationship here is exactly the kind of scope creep the pattern's threat model in Section 10.1 flagged.
- **Data-sensitivity consistency.** Does the node's classification align with the data-handling requirements Chapter 9 identified for PII and government ID data?
- **Infrastructure and vendor accountability.** Does the llm-provider node name a specific, approved provider and model — not a placeholder — and does the fluxnova-engine node's hosting environment carry the Common Cloud Controls mapping it needs? A CALM file

with a business-logic layer but no named infrastructure is exactly the gap this review exists to close.

This review sits with Hub Architecture and Governance Layer (MRM) jointly — the same pairing Chapter 7's Step 3 and Step 4 task lists already assign. Once both sign off, the CALM file is the approved architecture: not a proposal engineering will refine later, but the specification Chapter 11's Fluxnova process gets built against.

10.5 From CALM to Fluxnova: What Carries Forward

Every element defined in this chapter has a direct, named counterpart in Chapter 11. The two tool nodes — kyc-extraction-tool and sanctions-lookup-tool — become the two inner tools of the agentic ad-hoc subprocess in Section 11.3. The fluxnova-engine and fluxnova-dmn-service nodes are the literal CALM-level names for the workflow and decision-table components Chapter 11 builds out in full — the BPMN process and the DMN table are not new things Chapter 11 introduces, they're what these two nodes already committed the architecture to. The llm-provider node becomes the specific Provider and Model fields configured on the agentic ad-hoc subprocess in Section 11.2. The no-sanctions-override control becomes the row in the DMN decision table (Section 11.5) that routes any confirmed or possible match to escalation with no auto-clear path. The mandatory-human-review control becomes the analyst review task that Fluxnova's process definition will not let a case bypass. None of this is a loose thematic connection — Fluxnova's configuration literally references the node and control identifiers defined here, which is what makes the architecture review in Section 10.4 a real gate rather than a documentation exercise: get it wrong here, and Chapter 11 inherits the mistake.

Chapter 11 — Fluxnova in Depth: Modeling the KYC Process

Chapter 9 covered Step 1 of the nine-step path — codifying the KYC use case into a taxonomy classification, a risk shortlist, and a mitigation mapping, then contributing that artifact back to AIGF. This chapter picks up Step 5: turning those mitigations into an actual, running, enforced process. Fluxnova is where that happens, and because it's the component that ultimately determines whether the deterministic guarantees described in Chapter 3 hold in production, it deserves the same level of depth as Step 1.

11.1 From Mitigations to a Process Model

Recall the mitigations identified for the KYC use case in Chapter 9: mandatory human confirmation before any sanctions-adjacent case is cleared, a confidence threshold below which the agent must escalate rather than guess, and no ability for the agent to override a sanctions hit under any circumstance. Each of these is, in effect, a requirement for a specific enforcement point somewhere in the workflow. Fluxnova's job is to take that list of requirements and turn it into an explicit process definition — a BPMN diagram plus one or more DMN decision tables — where each mitigation corresponds to a concrete gateway, task boundary, or decision rule rather than an instruction the agent is merely asked to follow.

11.2 Two AI-Native Fluxnova Constructs

Everything in Section 11.1 could, in principle, be built with generic BPMN. What makes Fluxnova specifically suited to agentic workflows is two purpose-built constructs that connect the process layer directly to an LLM, without leaving the deterministic guarantees described in Chapter 3 at the door.

The agentic ad-hoc subprocess. This construct builds on Fluxnova 3.0's ad-hoc subprocess: a container of tasks with no fixed execution order. What makes the agentic variant different is who decides that order. At runtime, the engine queries an LLM; the LLM looks at the current business context and selects which of the subprocess's inner tasks to run, in whatever sequence and combination its reasoning calls for. Once selected, though, those tasks run exactly like any other workflow step — they are not a separate, informal side-channel the agent operates outside of, they are ordinary Fluxnova tasks with the engine's normal execution semantics.

Process authors configure the agent directly in the Fluxnova Modeler, not in code, which keeps the configuration itself reviewable alongside the rest of the process definition:

- **Provider and model** — which AI backend the subprocess calls.
- **System prompt** — the core goals and rules given to the agent for this subprocess.
- **Context variables** — process data passed securely into the agent's working context, so its reasoning is grounded in the actual case rather than generic instructions.
- **Restricted variables** — data explicitly held back from the model. An FSI organization could, for instance, pass a masked reference to a customer's identifier as a context variable while marking the underlying raw national ID number as restricted, so it never enters the prompt at all, even though the workflow itself has access to it.

Three properties of the construct matter most for governance, and they map directly onto the requirements in Section 11.1:

- **Safety sandbox.** The agent operates only within the boundary of the subprocess that was configured for it — it can select among the inner tasks defined there and nothing else, which is what turns "the agent needs to reason and act flexibly" and "the set of things it's permitted to do must be enumerable and enforced" from a tension into a design that satisfies both at once.
- **Full audit trail.** The engine records every action the agent takes, in the same way it records every other step in the process — there is no gap between what a human reviewer can see about a deterministic task and what they can see about an agent-selected one.
- **Enterprise reliability.** Standard error handling, retry logic, and persistence apply to agent-triggered tasks exactly as they would to any other task, so an agent's tool call failing mid-case behaves like any other transient failure the platform already knows how to recover from, not a special case engineers have to build around.

The MCP start event is a separate, complementary construct. Rather than triggering on a message, a timer, or a manual action, it exposes the entire workflow it begins as a callable MCP tool — turning the Fluxnova engine itself into an MCP server, with the workflow's expected input variables becoming that tool's parameter schema automatically. This matters for governance because a workflow already designed, reviewed, and enforced through everything described in this chapter doesn't need a separate, bespoke integration layer to be consumed elsewhere: it is already invocable as a standard MCP tool by another agent, another Fluxnova process, or a chat interface, with exactly the same boundaries intact regardless of who or what calls it.

The KYC process in this chapter uses the agentic ad-hoc subprocess directly

— Section 11.3 shows how. It doesn't currently use an MCP start event, since a document-received event is the natural trigger here, but the option is worth noting: if the FSI organization later wanted its case management system to invoke this same KYC review directly rather than waiting on a document upload, swapping the start event for an MCP start event would expose the whole reviewed, governed process as a tool that system could call, without touching anything downstream of the trigger.

Both constructs are active areas of open development within FINOS's Fluxnova projects — the underlying engine (fluxnova-bpm-platform), the modeler used to configure agents visually (fluxnova-modeler), and the AI-specific extensions (fluxnova-ai) each track this work as public GitHub issues and releases, consistent with the open-source, inspectable-by-anyone approach this guide argues for throughout.

11.3 Anatomy of a Fluxnova Process Definition

A Fluxnova process definition is built from a small set of BPMN primitives, each with a specific role:

- **Start and end events** (circles) — mark where a case enters and where it can terminate. A well-formed KYC process has exactly one start event and as many end events as there are distinct final outcomes.
- **Service tasks** — a discrete step with a defined input and output, such as an inner tool inside an agentic ad-hoc subprocess. Treated by the process model as a black box that never decides what happens next on its own.
- **User tasks** — a step that requires a human: the mandatory analyst review, the periodic sampling check. Fluxnova tracks these as open work items with an owner and a deadline, not a note in a log that a human was supposed to look at something.
- **Gateways** — the actual decision points, where the process branches based on a rule rather than the agent's judgment. In a well-designed KYC process, every gateway that matters for compliance is backed by a DMN decision table, not an inline condition buried in code.
- **Business rule tasks** — an explicit invocation of a DMN table, keeping the decision logic itself external, versioned, and independently reviewable, separate from both the process flow and the agent's reasoning.

11.4 The KYC Process, End to End

Putting those primitives together, the KYC document review process from Chapter 9 looks like this:

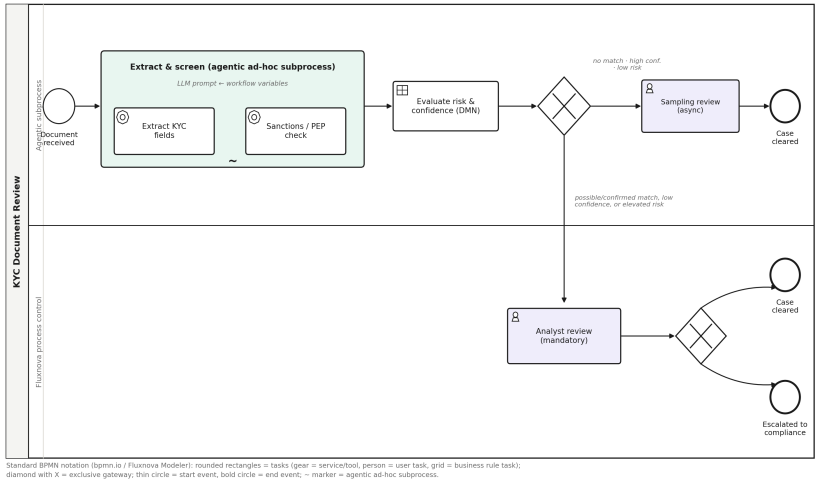


Figure 2. The Fluxnova process definition for KYC document review, in standard BPMN notation as Fluxnova Modeler renders it: rounded rectangles are tasks (gear icon = agent/tool, person icon = human, grid icon = a business rule task evaluating a DMN table), diamonds with an X are exclusive gateways, and the tilde marks the agentic ad-hoc subprocess.

Walking the diagram left to right: a case enters at document received and hands straight into the agentic ad-hoc subprocess. Inside it, the LLM receives a prompt parameterized with the case's workflow variables (the document reference, the case ID) and reasons about how to handle it — but the only actions available to it are the two inner tasks listed inside the subprocess boundary: extract KYC fields and check sanctions / PEP. It can call either tool, in either order, potentially more than once if its reasoning calls for it, but it cannot do anything else — there is no path from inside that boundary to any other system or action. Once the subprocess completes, it hands a fixed set of output variables (extracted fields, a confidence score, a match status) to the surrounding process, and the LLM's involvement in this case ends there. Those outputs feed a business rule task, evaluate risk & confidence, which doesn't consult the agent again — it looks up the outcome in a DMN table (Section 11.5), and the exclusive gateway immediately after it routes the case purely on that table's output.

Cases that come back with no sanctions match, high extraction confidence, and a low risk score take the top path: a sampling review, which is asynchronous and non-blocking — it doesn't hold up clearing the case, it exists to catch systemic drift across a sample of auto-cleared cases over time. Every other outcome — a possible match, low confidence, or an elevated risk score — routes to the bottom path: a mandatory analyst review that the process will not proceed past without a human decision. That decision it-

self is routed by a second exclusive gateway to one of two end states: case cleared, or escalated to compliance. Critically, a confirmed sanctions match can never reach the top path — the DMN table in Section 11.5 has no combination of inputs that produces an auto-clear outcome when the match field reads "confirmed," which is what makes the earlier claim in Chapter 9 ("cannot override a sanctions hit") a property of the process rather than a promise about the agent's behavior.

11.5 The DMN Decision Table

The first gateway's logic is fully externalized into a decision table, reviewable and testable independently of both the process flow and the underlying model:

Sanctions / PEP match	Extraction confidence	Risk band	Outcome
No match	High (≥ 95%)	Low	**Auto-clear → sampling review**
No match	High (≥ 95%)	Medium	**Escalate → analyst review**
No match	Below 95%	Any	**Escalate → analyst review**
Possible / weak match	Any	Any	**Escalate → analyst review**
Confirmed match	Any	Any	**Escalate → analyst review (cannot auto-clear)**

A few things worth noting about this table. It has no "agent recommends" column — the agent's outputs are the table's inputs, not a vote alongside the table's decision. The auto-clear outcome requires every one of three conditions to hold simultaneously (no match, high confidence, low risk); any single condition falling short routes to escalation by default, which means the table is deliberately biased toward the safer path rather than the efficient one. And because this table is a standalone artifact rather than logic embedded in a prompt, it can be unit-tested directly — feed it every combination of inputs and confirm the outcome column matches what compliance signed off on, something that is straightforward for a decision table and effectively impossible for a natural-language instruction.

11.6 Why This Satisfies the Determinism Requirement

This is the concrete version of the argument made abstractly in Chapter 3. The non-deterministic part of the system is scoped by construction: the agentic ad-hoc subprocess bounds the LLM's entire action space to the two

inner tools listed inside it, and its output to a fixed set of workflow variables. It cannot skip the gateway, cannot clear a case directly, and cannot see or influence the DMN table's outcome column — none of those are tools available inside the subprocess boundary, so none of them are things the model can reach no matter how it reasons. The deterministic part of the system — the gateways, the DMN table, the mandatory human task — is where every compliance-relevant decision actually happens, and every one of those components is a versioned, reviewable artifact rather than an instruction hoping the model complies. Run the same case through this process a thousand times with the same inputs, and it takes the same path a thousand times, which is precisely the property a regulator, an auditor, or an AIUC-1 adversarial tester needs to be true.

11.7 From Process Model to CI/CD and Observability

The process definition and its DMN tables don't stay in a design document — they become version-controlled artifacts that flow into the rest of the pipeline exactly as Chapter 1's diagram describes. The definition is validated in CI/CD alongside the rest of the codebase (Chapter 2), so a change to the DMN table's thresholds goes through the same review and automated checking as a code change — Chapter 12 shows exactly what that validation looks like for this use case. Once deployed, every execution of the process — which path a case took, which gateway fired, how long the analyst review sat open — is traced automatically, feeding the OpenTelemetry instrumentation and, ultimately, the evals that judge whether the deployed process is still behaving as designed (Chapter 14). If production data later shows the confidence threshold is miscalibrated — too many false escalations, or worse, a near-miss on the low-confidence path — that finding feeds back through observability into a revision of the DMN table, closing the loop back to policy definition just as Chapter 1 described, except now with a concrete mechanism for exactly what changes and how the change gets reviewed.

Chapter 12 — Validating Controls in CI/CD: The KYC Build Gate

Chapter 11 finished building the KYC process: a CALM specification, an agentic ad-hoc subprocess, and a DMN table that is never supposed to auto-clear a sanctions hit. None of that is worth anything if nobody actually checks it before the process reaches production. This chapter is Chapter 7's Step 6 — "embed compliance checks into CI/CD" — and Chapter 2.3's Controls stage, made concrete for the KYC use case specifically: the automated gate that stands between "we built something that should satisfy the risk assessment" and "we've actually confirmed it does."

12.1 Why This Step Exists Between Building and Running

Everything Chapters 9 through 11 produced is a claim: a claim that the CALM file's controls cover every mitigation in the risk shortlist, a claim that the agent's reachable surface matches what the taxonomy approved, a claim that the DMN table really does route every sanctions match to escalation. Claims are cheap to make and expensive to get wrong. The Controls stage exists to turn each of those claims into something mechanically checked before a human ever has to trust it — the same discipline Chapter 4's CI/CD integration already applies to every other change, applied here to the specific artifacts this use case produced.

12.2 What Gets Checked, and Against What

- **Structural validation of the CALM file.** Every relationship the Fluxnova process actually uses must be declared in the CALM specification, and nothing the CALM specification declares should go unused — the automated version of the "scope match" check Chapter 10.4 first introduced as a manual review step.

- **Control coverage.** Every mitigation in Chapter 9's risk shortlist must have a corresponding control in the CALM file — the automated version of Chapter 10.4's "coverage" check. A mitigation with no matching control fails the build rather than waiting for a reviewer to notice it's missing.

- **DMN table correctness.** The decision table from Chapter 11.5 gets exercised against every input combination that matters, with the outcome checked against what the linked control actually requires — this is the literal unit test Chapter 11.5 already argued the table's structure makes possible.

12.3 A Worked Example: Testing the KYC Build

What this looks like running, for the KYC use case specifically:

```
$ calm validate kyc-agentic-subprocess.calm.json
✓ All nodes declared (4 business nodes, 3 infrastructure nodes)
✓ All relationships reference declared nodes
✓ No undeclared relationship — agent's reachable surface matches
  the taxonomy classification from Chapter 9

$ calm controls check --against risk-shortlist.json
✓ no-sanctions-override      control present, linked to RI-KYC-07
✓ mandatory-human-review     control present, linked to RI-KYC-12
✓ token-budget-ceiling       control present
✓ shared-infra-allocation    control present
  4/4 mitigations from the Chapter 9 risk shortlist have a matching
  control

$ fluxnova dmn-test kyc-risk-confidence.dmn --cases dmn-test-cases.json
✓ no match, high confidence, low risk        → auto-clear   PASS
✓ no match, high confidence, medium risk     → escalate     PASS
✓ no match, low confidence, any risk         → escalate     PASS
✓ possible match, any confidence, any risk   → escalate     PASS
✓ confirmed match, any confidence, any risk  → escalate     PASS
                                                    (cannot auto-clear)
  5/5 test cases passed — 0 combinations produce an auto-clear
  outcome on a sanctions match

BUILD: PASSED — deployment approved
```

The last line of that test matrix is the one worth sitting with. Chapter 9 asserted that a confirmed sanctions match can never resolve to auto-clear; Chapter 11 built a DMN table designed to guarantee it; this is where that guarantee stops being a design intention and becomes a passing test, re-run automatically on every future change to the table, the CALM file, or the risk shortlist that feeds them both.

12.4 What Happens on Failure

Any one of these checks failing blocks the build the same way a failing unit test blocks any other software change — the pull request from Chapter 9.4 cannot merge, and the use case cannot progress to Chapter 7's Step 7 deployment. There's no special-casing between a KYC-specific control and a general one drawn from the AIGF catalog: the CI/CD integration treats them identically, which is exactly Chapter 4.1's point about compliance checks running where code already runs. Ownership follows Chapter

7.1's structure directly — Hub MLOps/DevSecOps maintains the pipeline mechanics that run these checks, while the Governance Layer (MRM) is who defines what "pass" actually means for a given control, particularly the DMN test cases a change might need to satisfy.

12.5 From CI/CD Gate to Deployment

Once every check passes, what gets deployed is the exact bundle this chapter validated — the CALM specification and the Fluxnova process definition, not a hand-verified approximation of them. But "what passed the gate" and "what's actually running" are two different claims, and everything so far in this chapter only establishes the first one. Section 12.6 closes that gap.

12.6 Runtime Conformance: A Control Plane for What's Deployed

Every check in Sections 12.2 through 12.4 runs before merge. That's deliberate — catching a missing control before it ships is far cheaper than catching it after — but it also means those checks are silent about two entirely realistic failure modes: a deployment that doesn't match what was validated, and a live system that drifts away from its validated state sometime after it started running. Neither is hypothetical. A hotfix applied directly to a running Fluxnova engine, a manually edited tool binding, an infrastructure change made outside the normal pipeline — any of these leave the CI/CD gate from Sections 12.2 through 12.4 with nothing to say, because none of them go through it. Closing that gap needs a control plane: something that keeps checking conformance continuously, not just at the moment of merge.

- **Sign the validated bundle, verify the signature at deploy.** Once the CALM specification and Fluxnova process definition pass Section 12.3's checks, hash and sign that exact bundle as a build artifact — the same supply-chain attestation pattern (SLSA, in-toto) used to sign container images. The deployment step then refuses to run anything whose signature doesn't match what was actually validated, which is what stops a hand-edited config from ever reaching production labeled as if it had passed the gate.
- **Diff declared state against live state.** CALM's own tooling already has the mechanism for this: calm-cli ships a diff command that compares two CALM documents — or two moments in a CALM timeline — matching nodes and relationships by their unique identifiers and classifying each change as added, removed, modified, or renamed. Exporting the live Fluxnova engine's actual configuration back into CALM's document format and running it against the git-committed,

validated version turns that comparison into a drift check: any difference is something Section 12.3's pre-merge checks couldn't have caught, because it happened after deployment.

- **Make the diff continuous, not a one-off audit.** A single comparison answers "is it conformant right now." A control plane runs that same comparison on a schedule, or reacts to change events, the way GitOps controllers like ArgoCD or Flux continuously reconcile a Kubernetes cluster against its declared manifests. The KYC use case's CALM file in git is the source of truth; the controller's job is noticing, promptly, whenever the running system stops matching it — and either alerting the Governance Layer or reverting the unauthorized change automatically, depending on how much autonomy the control itself is configured to have.

- **Cross-check declared boundaries against observed behavior.** Chapter 14's observability layer already tags every execution trace with the CALM node and control identifiers involved. That's a second, independent conformance signal, orthogonal to comparing configurations: if any trace ever shows the agentic subprocess invoking a tool call outside its declared relationships, that's drift proven from actual behavior, not from inspecting a config file that might not reflect what's really executing.

- **Push the same validation to the deployment admission point, not just CI.** The checks from Section 12.2 can run a second time as an admission gate on the deployment side itself — the same pattern Kubernetes' OPA/Gatekeeper uses to block a resource that doesn't satisfy policy, applied here to a Fluxnova process definition. This is the backstop for the one scenario none of the above fully covers: a change that skips the normal CI/CD path entirely. An admission gate at the one chokepoint everything has to pass through to actually run catches it regardless of how it got there.

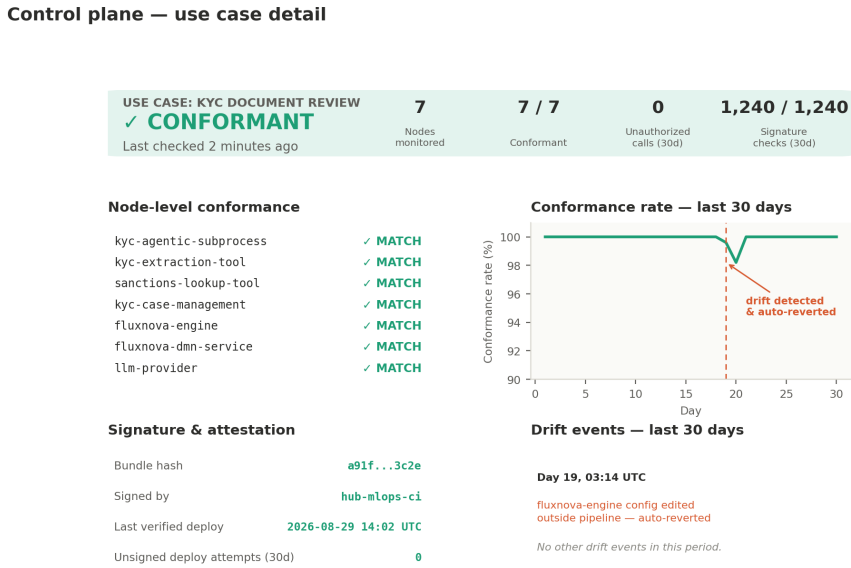


Figure 7. An illustrative control plane dashboard for the KYC agentic subprocess: overall conformance status, node-by-node comparison against the declared CALM file, signature verification, and a 30-day history showing one drift event caught and auto-reverted.

Every panel in that dashboard maps to one of the five mechanisms above: the top banner is the single unambiguous status a Governance Layer reviewer actually wants first; the node table is this section's diff, run against every CALM node rather than summarized away; the signature panel is the attestation check from the first bullet; the trend line is what makes the reconciliation loop's history visible rather than only its current state; and the drift log is where a caught-and-reverted change becomes a recorded event instead of a silent non-issue. The one drift event in this illustration — a fluxnova-engine config edited outside the pipeline — is exactly the failure mode this section opened with, shown here as something the control plane actually caught rather than a hypothetical risk.

This screen is scoped to a single use case on purpose — Chapter 15 extends the same control plane with a portfolio-wide home screen and a dedicated tokenomics view, so a use case's conformance status is one tile in a larger picture rather than a standalone tool.

None of this replaces Sections 12.2 through 12.4 — it's the continuous complement to a one-time gate. The CI/CD checks answer "does this pass before it ships." The control plane answers "is it still true now," for as long as the KYC process keeps running. Together, they're what makes Chapter 9's original claim — that a confirmed sanctions match can never auto-clear — something an FSI organization can stand behind on any given day, not

just on the day the pull request was reviewed.

Chapter 13 — Evals: Measuring What Determinism Doesn't Cover

"Evals" has been sitting in this guide's own pipeline diagram since Chapter 1 — listed alongside OpenTelemetry and dashboards in the Observability & Feedback stage, and referenced again in Chapters 2, 12, and 14 as the thing that judges whether a deployed process is "still behaving as designed." None of those mentions ever said what an eval actually is, or how to build one. This chapter fixes that.

13.1 What Determinism Doesn't Cover

Chapter 3 drew a hard line: Fluxnova makes the process deterministic, even though the LLM inside it isn't. That line is real and it matters, but it answers a narrower question than it sounds like. Chapter 11's agentic ad-hoc subprocess guarantees the extraction tool and the sanctions lookup are the only two things the agent can do — it says nothing about whether the extraction is any good. A well-designed CALM control (Chapter 10) stops the agent from calling a tool it isn't supposed to; it can't tell you whether that tool's output actually got the applicant's date of birth right. Evals are how a team answers that second, harder question — the quality and safety of what happens inside the boundary Fluxnova already drew, not whether the boundary held.

13.2 Eval-Driven Development: The Methodology

Eval-driven development (EDD) is a top-down approach: define the standard a good agent output looks like before building toward it, the way test-driven development defines a test before writing the code that passes it. As described by DeepEval, an open-source evaluation framework built around this workflow, EDD is a three-step loop:

- **Curate a small, high-quality dataset** — around 100 "goldens" (input/expected-output pairs), not thousands of auto-generated ones.
- **Define a handful of metrics that actually correlate with performance** — typically 3 to 5, not an exhaustive checklist.
- **Iterate until every metric passes** — treating a failing eval the same way a failing test blocks a merge.

The methodology borrows its shape from test-driven development, but the underlying evaluations behave nothing like a normal unit test, which is exactly why EDD needs a discipline of its own rather than just running more tests. Where a TDD assertion is free, instant, and objectively pass/fail, an

eval is often an LLM judging another LLM's output — it costs money to run, takes real time, and produces a subjective, graded score rather than a clean true/false. That's precisely why the dataset stays small and curated rather than exhaustive: a bank running thousands of low-quality, auto-generated test cases through an LLM-as-judge metric is paying real money to learn very little, where a hundred carefully chosen cases, reviewed by someone who understands the use case, actually tell a team something.

13.3 What to Evaluate: Security and Functionality

Two categories cover most of what matters, and both map directly onto risks Chapter 9's heuristic assessment already names rather than introducing anything new:

- **Security** — prompt injection, PII leakage, toxicity, jailbreaks. For the KYC extraction tool specifically, this is the empirical test of exactly the risk Chapter 9's shortlist flagged: can a maliciously crafted document manipulate the agent's reasoning, or cause it to leak data it shouldn't.
- **Functionality** — task completion, tool-calling accuracy, and factual correctness. For extraction, this is whether the fields it pulls out of a document are actually right, not merely well-formatted; for the sanctions lookup, whether the agent calls the tool with the correct, unmodified identity details.

Security evals are close to non-negotiable — an agent that leaks PII or can be steered off-task by an injected instruction is a governance failure regardless of how well it performs otherwise. Functionality evals are where a team's actual understanding of what "good" looks like for their specific use case has to show up; nobody outside the team building the KYC extraction tool can hand them the right metric for it.

13.4 Unit Evals vs. Regression Evals

This is the same unit-versus-integration distinction Chapter 12 already applies to structural checks, carried over to behavioral ones:

- **Unit evals** score one component in isolation — the extraction tool's output against a document, independent of what happens afterward in the process. This is the direct analogue of validating a single CALM node rather than the whole specification.
- **Regression evals** run end-to-end over the full curated dataset every time the prompt, the model, or the subprocess's configuration changes — the behavioral equivalent of Chapter 12's full DMN test matrix, run on every change rather than once.

Both matter, for the same reason Chapter 12 argues both structural checks matter: a unit eval catches a regression in one tool before it's masked by everything downstream; a regression eval catches an interaction between components that no single unit eval would ever see on its own.

13.5 A Worked Example: Evals for the KYC Extraction Tool

A small golden set for the extraction tool from Chapter 11 might look like this — a handful of representative documents, each with a known-correct extraction, deliberately including the edge cases a team already knows are hard:

```
GOLDEN SET — kyc-extraction-tool (excerpt, 3 of 104 goldens)

golden_001  input: US passport, standard MRZ format
            expected: { dob: "1985-03-14", name: "J. RIVERA", doc_id: "..." }

golden_047  input: UK driving licence, DD/MM/YYYY date format
            expected: { dob: "1990-11-02", name: "A. KHAN", doc_id: "..." }

golden_089  input: scanned corporate registry extract, low image quality
            expected: { entity_name: "...", registration_no: "...",
  jurisdiction: "..." }
```

Three metrics carry most of the signal for this tool: a Task Completion metric (did the extraction produce a structurally valid, complete output), a custom field-accuracy metric scored against the golden's expected values, and a PII-leakage check confirming the agent's reasoning trace never echoes sensitive fields anywhere it shouldn't. Running that suite looks like this:

```
$ deepeval test run test_kyc_extraction.py

test_kyc_extraction.py::test_extraction[golden_001] PASSED
  Task Completion:        0.94   (threshold 0.85)   PASS
  Field Accuracy (G-Eval): 0.91   (threshold 0.85)   PASS
  PII Leakage:             0.00   (threshold 0.05)   PASS

test_kyc_extraction.py::test_extraction[golden_047] FAILED
  Task Completion:        0.78   (threshold 0.85)   FAIL
  Field Accuracy (G-Eval): 0.69   (threshold 0.85)   FAIL
  PII Leakage:             0.00   (threshold 0.05)   PASS
  Reason: extracted date of birth transposed day/month on a
          non-US date format document

====================== 102 passed, 2 failed in 41.3s ======================
BUILD: FAILED — 2 goldens below threshold, merge blocked
```

That failure is a genuinely useful one to catch here rather than in production: it's exactly the kind of narrow, format-specific bug a manual code review would likely miss, and exactly the kind of thing 4,820 real KYC cases a month (Chapter 12's volume) would eventually surface anyway — just later, and after some number of UK-format documents had already been misread.

13.6 Evals as a CI/CD Gate

This slots into Chapter 12's build gate as a third check, alongside CALM structural validation and the DMN test matrix — not a separate pipeline, the same one:

```
$ calm validate kyc-agentic-subprocess.calm.json
✓ All nodes declared, all relationships reference declared nodes

$ fluxnova dmn-test kyc-risk-confidence.dmn --cases dmn-test-cases.json
✓ 5/5 test cases passed

$ deepeval test run test_kyc_extraction.py
✗ 102/104 goldens passed — 2 below threshold

BUILD: FAILED — fix failing evals or update the golden set before merging
```

One thing worth being precise about: this isn't a native Fluxnova feature, and this guide isn't claiming FINOS and DeepEval have an official integration. DeepEval instruments at the LLM call itself — wrapping the Anthropic or OpenAI client the agentic ad-hoc subprocess calls, or the surrounding function, with a lightweight decorator — which makes it independent of whatever orchestrates the call around it. That's precisely why it fits here without requiring one: Fluxnova bounds where the LLM can act (Chapter 11), and DeepEval evaluates what it does once it acts, and the two never need to know about each other to both be true at once. Ownership follows the same split Chapter 7.1 already established for other CI/CD checks — Hub MLOps/DevSecOps runs the pipeline mechanics, while the Business Spoke's AI engineers own the goldens and the metrics that matter for their specific use case, per Section 13.3's point that nobody outside the team can hand them that list.

13.7 From One-Time Gate to Continuous Signal

Chapter 14's observability layer already tags every production trace and mentions evals as part of what closes the loop; this is what actually fills that in. The same regression suite that blocks a merge in Section 13.6 also runs on a schedule against a sample of real production traces, not just the curated golden set — catching the drift Chapter 14.6 describes (an escalation rate climbing, or in this case, an extraction accuracy quietly declining) before it shows up as a compliance incident rather than after. A finding here feeds back exactly where Chapter 7's Step 8 already says it should: a metric trending down triggers the same feedback-loop investigation as a drifting escalation rate, because both are the same kind of signal — evidence that something approved a while ago needs a second look now.

This is also, finally, what closes the loop Chapter 1 opened on page one. The Observability & Feedback stage was always going to include evals — this chapter is what makes that entry in the diagram mean something concrete rather than a term left for later.

Chapter 14 — Observability in Depth: Evidence, Metrics, and the KYC Process in Production

Chapter 11 ended at the point of deployment, with a one-paragraph promise: every execution gets traced, feeding OpenTelemetry and, eventually, the evals that judge whether the system still behaves as designed. This chapter makes good on that promise in the same depth as Chapters 9 through 11 gave the rest of the build — what observability actually needs to capture once the KYC process is live, how it turns into evidence a regulator or an auditor can use, and how it gives engineering and Governance a shared, factual view of what the deployed system is actually doing.

14.1 Two Audiences, One Signal

Observability is often framed as an engineering concern — is the system up, is it fast, is it erroring. That's real, but it's only half of what this pipeline needs. The other half is Governance's question from Chapter 5: not just is it working, but can we prove the controls held. The value of building observability on top of Fluxnova's execution traces (Chapter 11) rather than as a separate bolt-on is that both questions get answered from the same underlying data. An engineer watching latency and an MRM analyst confirming that a sanctions match never auto-cleared are querying the same trace store, not two disconnected systems that happen to sit next to each other.

14.2 The Three Signals: Traces, Metrics, and Logs

OpenTelemetry, introduced in Chapter 1's pipeline diagram, organizes observability data into three kinds of signal, and the KYC process produces all three:

- **Traces** capture the full execution path of a single case: which tasks ran, in what order, which gateway fired, and how long each step took. A trace is the answer to "what happened to this specific case" — exactly the granularity Chapter 11's determinism argument depends on being reconstructable.
- **Metrics** aggregate traces into counts and rates across many cases: how many times a task ran, what fraction succeeded, how long it typically takes. Metrics are what turn thousands of individual traces into a small number of numbers a dashboard can show at a glance.
- **Logs** capture event-level detail inside a step — a tool call's input and output, an error message, a retry. Logs are where an engineer goes

after metrics or a trace has pointed at a specific case or task that needs a closer look.

None of this is specific to agentic systems — it's the same three-signal model any well-instrumented service uses. What makes it relevant to governance is what gets tagged onto each signal: a trace or a log line associated with a Fluxnova execution carries the CALM node and control identifiers from Chapter 10, so a query for "every case where the no-sanctions-override control was evaluated" is a normal query against normal observability data, not a special-purpose audit system.

14.3 Task Executions: Success Rate, Duration, and Invocation Counts

At the metric level, every task in the KYC process — the two agent tool calls, the DMN gateway, the two human review tasks — gets tracked the same way: how often it runs, how often it succeeds, and how long it takes. This is the concrete, per-task view of the process Chapter 11 modeled:

Task	Invocations (30d)	Success rate	Avg. duration	P95 duration
Extract KYC fields (tool)	4,820	98.7%	2.1s	4.8s
Sanctions / PEP lookup (tool)	4,820	99.9%	0.8s	1.6s
Risk & confidence gateway (DMN)	4,820	100.0%	0.02s	0.04s
Sampling review (human, async)	3,960	100.0%	1.4 days	3.1 days
Analyst review (human, mandatory)	860	100.0%	6.2 hours	1.8 days

A few things worth reading off this table. The two agent tool calls run on every case (4,820 invocations, matching total case volume) and complete in seconds, which is expected for a bounded tool call. The sampling review and analyst review run far less often — 3,960 and 860 times respectively — because they're the two branches of the gateway from Chapter 11, and their durations are measured in hours and days rather than seconds, because they wait on a human. A success rate below 100% for the agent tasks (98.7% for extraction) isn't itself alarming — it reflects retried failures like a malformed upload — but a sustained drop would be exactly the kind of signal that triggers the review Section 14.6 describes. The two human tasks show 100% success by construction: Fluxnova doesn't record a human task as complete until a reviewer has actually recorded a decision, so there's no partial or malformed outcome to measure.

This is also where the agentic ad-hoc subprocess's tool-bounded design (Chapter 11) pays off for observability specifically. Because the LLM's only actions are the two listed tools, every invocation the metrics table counts is a tool call this table already expected — there's no category of "the agent did something else" to account for, because the subprocess boundary doesn't allow it.

14.4 Evidence of Control Usage: Proving the Mitigations Held

Metrics on tasks answer an operational question. This section answers Chapter 5's compliance question directly: not just that the process ran, but that the specific controls from Chapter 10 actually did what they were supposed to, every time. Because each control carries its identifier through the CALM specification and into the DMN table (Chapter 11), the evidence is a straightforward query against the same trace data — how many times was this control evaluated, how many times did it trigger, and did the outcome match what the control requires:

Control	Times evaluated	Times triggered	Verified outcome
no-sanctions-override	4,820	142 (confirmed or possible match)	100% routed to escalation — zero auto-clears on a match
mandatory-human-review	4,820	860 (below confidence or risk threshold)	100% held for analyst review before any outcome recorded

This table is the literal answer to the claim Chapter 9 made when it said a confirmed sanctions match can never reach an auto-clear outcome. It's not an assertion about the DMN table's design anymore — it's a count, over 4,820 real cases, confirming that every one of the 142 cases where the control triggered was actually routed to escalation. This is precisely the kind of evidence Chapter 6 described AIUC-1's adversarial testing and audit process wanting to see, and precisely what an internal audit or a regulator examining the KYC process would ask for first.

14.5 Dashboards for Different Audiences

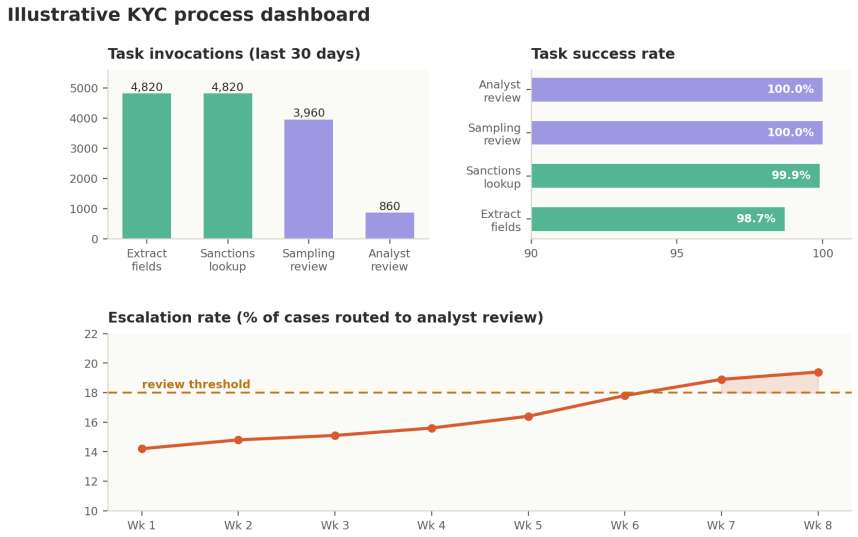


Figure 5. An illustrative KYC process dashboard: task-level invocation counts and success rates alongside a compliance-relevant trend — the escalation rate crossing a review threshold over eight weeks.

Chapter 1's pipeline diagram showed dashboards as a single node with two views — operational and compliance — and this figure shows both at once, because in practice they're built from the same data and often live on the same screen. The top two panels are the operational view: how much volume is moving through each task, and how reliably. The bottom panel is the compliance-relevant one: the escalation rate — the share of cases routed to a human rather than auto-cleared — tracked over time against a review threshold Governance set in advance.

The trend in that bottom panel is deliberately drawn crossing the threshold, because that's the scenario Section 14.6 exists to describe. A rising escalation rate isn't automatically a problem — it might mean the DMN table's thresholds from Chapter 11 are working exactly as intended against a genuinely riskier mix of cases. But it's also exactly the kind of drift Chapter 11's closing paragraph warned about, and the dashboard's job is to surface it as a visible trend well before it becomes a compliance incident, not to render a verdict on its own.

14.6 From Observability to the Feedback Loop

This is Step 8 from Chapter 7, made concrete. When the escalation rate in Section 14.5 crosses its threshold, that's not the end of the story — it's the trigger for exactly the review Chapter 7's Step 8 task list assigns to the Gov-

ernance Layer, with Business Spoke AI engineering support. The investigation starts with the same data this chapter has already described: pull the traces for cases escalated in the weeks the rate climbed, check whether they cluster around a specific input pattern the original risk assessment (Chapter 9) didn't anticipate, and decide whether the finding changes the DMN table's thresholds (Chapter 11), the risk catalog (Chapter 9), or both.

Whatever the finding, it feeds back into the same artifacts this guide has built throughout: a revised DMN table gets the same CI/CD-enforced review any other change gets (Chapter 4), a revised risk mapping updates the CALM controls it's linked to (Chapter 10), and if the finding is significant enough to matter beyond this one use case, it's a candidate for contribution back to AIGF itself, exactly as Chapter 9 described for the original KYC pull request. This is also, concretely, the evidence base Chapter 7's Step 9 and Chapter 6's AIUC-1 discussion both depend on: an FSI organization deciding whether to pursue external certification, or preparing for a quarterly re-audit, is drawing on the same task metrics and control evidence tables this chapter just walked through — not compiling them for the first time under deadline pressure.

14.7 Generating Compliance Reports: Per-Case and Aggregate

Everything this chapter has described — traces, task metrics, control evidence — is queryable data, which means a compliance report isn't a document someone writes; it's a document the pipeline assembles on request. Two report shapes cover most of what Governance actually needs, and both are generated from the same underlying trace store described in Section 14.2, not compiled by hand from separate sources.

A single-case report

The narrowest and most common request is: show me exactly what happened on this one case — for a customer complaint, a spot-check from an examiner, or a follow-up on a specific escalation. Because every task, gateway, and control evaluation in the KYC process carries the identifiers from Chapters 10 and 11, this report is a direct query against a single trace, not a reconstruction:

```
KYC CASE COMPLIANCE REPORT
Case ID: KYC-2026-0483217
Generated: 2026-08-31 14:32 UTC (on demand)

USE CASE        KYC document review (agentic ad-hoc subprocess)
CALM SPEC       kyc-agentic-subprocess v3.2
```

```
FLUXNOVA PROC   kyc-review-v3 (deployed 2026-06-11)

EXECUTION PATH
  1. Document received                      14:00:02
  2. Extract KYC fields (tool)               14:00:04              success,
  confidence 0.97
  3. Sanctions / PEP lookup (tool)           14:00:05              success,
  match: none
  4. Risk & confidence gateway (DMN)         14:00:05              outcome:
  escalate (risk band: medium)
  5. Analyst review (human, mandatory)       14:00:05 → 18:42:11
  assigned: J. Alvarez
  6. Case cleared                            18:42:11

CONTROLS EVALUATED
  no-sanctions-override     evaluated: yes  triggered: no   outcome: n/a, no
  match found
  mandatory-human-review    evaluated: yes  triggered: yes  outcome: held for
  review;
                                                             cleared by
  analyst

AUDIT TRAIL
  Trace ID: 7fae2c1e-4b9a-4e2d-8f13-2c5a9b31d0e7
  Retained per data retention policy: 7 years
```

Notice what this report doesn't require: nobody went back through logs to reconstruct what happened, and nobody had to explain the DMN table's logic from memory. Every line is a direct readout of data the process generated as it ran — which is the same "evidence as a byproduct, not a project" point the Introduction and Chapter 6 both make, now shown as an actual artifact rather than an argument.

An aggregate report, across all instances

The second shape answers a broader question — not what happened on one case, but how the process behaved as a whole over a period — the kind of report a monthly Governance review, a board update, or an AIUC-1 quarterly re-audit (Chapter 6) actually needs. It's assembled from the same data as the single-case report, aggregated rather than filtered to one trace, and it draws directly on the tables and figure already built in this chapter:

- **Executive summary** — period covered, total case volume, and the headline numbers a reader wants first: overall success rate, escalation rate, and whether either moved outside its expected range.

- **Task-level performance** — the same shape as Section 14.3's table, for the period in question: invocation counts, success rates, and durations for every task in the process.
- **Control evidence** — the same shape as Section 14.4's table: how many times each control was evaluated, how many times it triggered, and confirmation that every triggered instance resolved the way the control requires.
- **Trend and anomalies** — built from the same dashboard in Section 14.5: whether the escalation rate crossed its review threshold during the period, and if so, a note on whether Section 14.6's investigation was opened and what it found.
- **Certification readiness** — for a use case where Chapter 7's Step 9 decision was to pursue AIUC-1, a note on whether this period's evidence is consistent with the certification already granted, or whether drift noted above warrants flagging ahead of the next quarterly re-audit.

Both reports exist for the same reason: because the underlying data is structured and queryable from the moment the process runs, generating either one is a matter of asking the right question of data that already exists — not a research project that starts when someone asks for it.

Chapter 15 — Tokenomics: Attribution, Cost, and Value

Chapter 14 gave every task in the KYC process a cost dimension almost in passing — the observability layer's job, per Chapter 1's diagram, includes cost tracking alongside behavior. This chapter takes that dimension seriously on its own terms. Tokenomics is not the same question as "how many tokens did we spend this month" — that number, on its own, tells an FSI organization almost nothing useful. The real questions are who spent them, on what, and whether what they bought was worth more than it cost.

15.1 Tokenomics Is an Attribution Problem, Not a Metering Problem

A single aggregate token bill is a finance line item, not a governance signal. It can't tell a CAIO whether the Financial Crime & Fraud spoke's spend is concentrated in one runaway use case or spread evenly across five healthy ones, whether a given agent's cost per case is trending up because of genuine complexity or a badly-tuned prompt, or whether the institution is spending more on a use case than that use case is worth. Getting from a bill to an answer requires attributing every token to three things at once: the specific agent that spent it, the specific use case it ran inside, and the specific department that owns that use case. Miss any one of those three and the number is close to meaningless — a department-level total with no use-case breakdown hides a single expensive outlier; a use-case total with no agent breakdown hides which specific tool call or prompt is driving cost.

This isn't a problem an FSI organization needs to solve alone or from a blank page. In August 2026, the Linux Foundation launched the Tokenomics Foundation — a vendor-neutral body, backed by JPMorganChase, BNY, SAP, ServiceNow, IBM, and two dozen others, building open standards for exactly this: shared definitions for token value and density, a reference model for the full cost of AI, and standardized cost telemetry contributed into FOCUS, the FinOps Foundation's existing cost-and-usage specification. The relationship to this guide's argument is direct: the same case Chapter 2 makes for AIGF over a proprietary risk framework applies here — an FSI organization's internal tokenomics practice should align to this emerging open standard as it matures, rather than inventing its own token taxonomy the way it shouldn't invent its own risk taxonomy.

One caution worth setting expectations with early: falling per-token prices do not mean falling total AI spend. Industry analysis has projected inference costs continuing to drop through the rest of the decade, and enterprise AI spend has continued rising anyway — usage and workload complexity grow faster than efficiency gains reduce unit cost. A CAIO reporting to a

board should expect this pattern rather than be caught by it: the goal of tokenomics is not to make the bill shrink, it's to make sure the bill's growth stays tied to growing value.

15.2 Three Axes of Attribution: Agent, Use Case, Department

- **By agent.** Within a single use case, different agentic subprocesses — or different tool calls inside the same subprocess — consume tokens differently. In the KYC example, the agentic ad-hoc subprocess's own reasoning, its call to the extraction tool, and its call to the sanctions lookup tool are three distinguishable consumers, not one lump sum, because each is already a separately identified node in the CALM specification from Chapter 10.
- **By use case.** Aggregating across every instance of one process — every KYC case run in a month — gives the number a Business Spoke actually manages against: what does running this use case cost, in total and per case, and is that trending in a direction anyone approved.
- **By department.** Aggregating across every use case a given Business Spoke runs — not just KYC, but every agent Financial Crime & Fraud operates — gives the Hub and the Governance Layer the number that actually matters for budget and prioritization conversations: which parts of the hub-and-spoke structure from Chapter 7 are consuming the shared model infrastructure, and how that maps to the value each of them is producing.

None of these three views replaces the others. An FSI organization that only tracks department-level spend can't see that one agent inside Financial Crime & Fraud is responsible for the department's entire increase; an FSI organization that only tracks per-agent spend can't roll that up into a departmental budget conversation. The three axes are the same underlying data, sliced three ways.

15.3 Tagging Consumption at the Source

This isn't a separate tracking system bolted on after the fact — it's the same tagging mechanism Chapter 14 already described for control evidence, extended to carry one more field. Every CALM node from Chapter 10 already has a unique identifier; every Fluxnova trace from Chapter 11 already carries that identifier through execution. Attributing tokens by agent is a matter of recording token counts against the same kyc-agentic-subprocess, kyc-extraction-tool, and sanctions-lookup-tool identifiers already in use — not inventing a new namespace. Attributing by use case and department follows the same path one level up: the use case's taxonomy classification

from Chapter 9 already names which Business Spoke owns it, so a token count tagged to a use case is already, transitively, tagged to a department. The practical implication is that tokenomics doesn't require new infrastructure so much as one additional field on data the pipeline is already generating.

15.4 Beyond Tokens: The Full Cost of AI

Token counts are the easiest number to get and the most incomplete one to stop at. Industry cost analysis of enterprise AI deployments consistently finds that compute and inference are only about half of total AI spend — the rest is storage, networking, power and facilities, and the operational labor of running the platform. A tokenomics practice that only ever reports token counts is, in effect, reporting roughly half the bill. The full picture has three layers, and the KYC use case already has a named node for each of the first two, from Chapter 10's CALM specification:

- **Token / inference cost** — the llm-provider node: a near-pure marginal cost that scales directly with case volume, exactly what Sections 13.1 through 13.3 already attribute.

- **Orchestration cost** — the fluxnova-engine node. Fluxnova itself is license-free, open-source software — but running it is not free. Someone hosts the compute and storage the engine runs on, someone (Hub MLOps/DevSecOps, per Chapter 7.1) keeps it patched and scaled, and an FSI organization may choose to pay a FINOS-affiliated service provider (Chapter 8.2) for a support contract. Unlike token cost, this is closer to a fixed cost: it scales with infrastructure capacity, not with how many KYC cases ran this month.

- **Operational / oversight cost** — the human review time already in Chapter 14's task metrics table: analyst hours spent on escalated cases.

The fixed-versus-marginal distinction between the first two layers matters for a reason beyond precision. Because Fluxnova is Hub-owned, shared infrastructure — the same "build once, reuse across every use case" asset Chapter 7 already describes for the taxonomy and reference architectures — its hosting and support cost is naturally amortized across every use case running on it, not paid fresh by each one. That's a second, cost-side version of Chapter 7's argument that an FSI organization's second and third use cases move faster than its first: they're also cheaper per case, because they share infrastructure whose cost is already sunk.

The buy-versus-host decision

The llm-provider node itself represents a choice worth surfacing explicitly rather than leaving implicit in a configuration field: whether the FSI organization calls a model through a metered API, or hosts and serves its own models on owned infrastructure. Industry TCO modeling finds a real inflection point in this decision — below roughly 70 to 85 billion tokens a year, pay-as-you-go API access tends to be more cost-effective; above it, self-hosted infrastructure's fixed costs are spread over enough volume to win out, at the price of the capital investment and operational maturity self-hosting demands. This isn't a decision to make once and forget — it's exactly the kind of architectural choice Chapter 10's architecture review (Section 10.4) should revisit periodically as a use case's volume grows, the same way Chapter 7's Step 8 revisits a DMN table's thresholds as production evidence accumulates.

15.5 From Consumption to Value: The Net Benefit Equation

Attribution and full-cost accounting answer what something costs. They don't answer whether it was worth running. That question needs a business value figure to set against the cost, and the comparison that actually matters isn't the AI system's cost in isolation — it's the AI system's cost against what the same work would have cost the traditional way. Stated plainly:

- **Net AI value = (cost of the traditional approach) − (full cost of the AI-assisted approach).**

The full cost of the AI-assisted approach is now Section 15.4's three layers together: token cost, amortized orchestration cost, and remaining human oversight cost. The cost of the traditional approach is what fully manual review of the same volume would have cost in analyst time alone. A second, complementary value metric is worth tracking alongside the dollar figure — the Tokenomics Foundation's own roadmap proposes measuring AI value starting with the share of work completed without human involvement, set against what that work costs today. The KYC process already has this number: Chapter 14's escalation rate means roughly 82.2% of cases clear without ever reaching a human reviewer. That figure and the dollar-based net benefit are two views of the same underlying success, and an FSI organization reporting both gives a board a number it can also compare against how other institutions and other use cases are doing, once the Tokenomics Foundation's benchmarks mature.

15.6 A Worked Example: KYC Tokenomics

Putting real illustrative numbers against the framework, using the same case volume Chapter 14 worked with (4,820 cases over 30 days, roughly 17.8% escalated to analyst review):

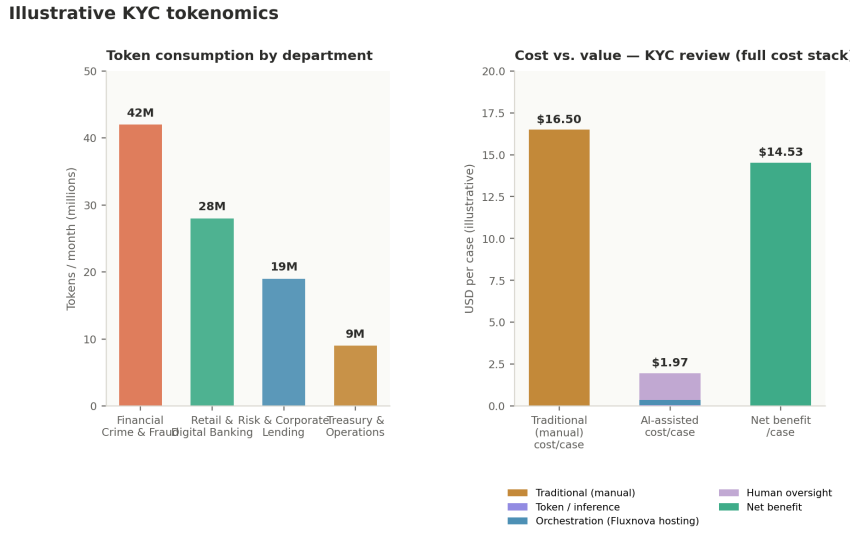


Figure 6. Illustrative KYC tokenomics: token consumption attributed by department (left), and the full cost stack — token, orchestration, and human oversight — set against the traditional approach and the resulting net benefit (right).

The left panel is Section 15.2's department axis made concrete — Financial Crime & Fraud, where the KYC use case lives, carries the largest share of token consumption in this illustration, but the breakdown lets the Hub see immediately that this isn't spread evenly, and lets Governance ask a specific, answerable question if that concentration looks wrong rather than an unanswerable one about a single combined total.

The right panel is Section 15.5's equation worked through with the full cost stack from Section 15.4, not tokens alone. A fully manual KYC review, at roughly 22 minutes of analyst time per case, costs approximately $16.50 per case in analyst time alone. The AI-assisted process costs a small fraction of a cent per case in tokens, roughly $0.30 per case in amortized Fluxnova hosting and support, and analyst time only for the 17.8% of cases that reach mandatory review — blending to roughly $1.97 per case across the full volume once every layer is counted. The net benefit, per case, is the difference: approximately $14.53 — modestly lower than a token-only estimate would suggest, but a more defensible number, since it doesn't quietly omit real infrastructure cost the way a token-only figure would. At 4,820 cases a month, that's a meaningful monthly figure an FSI organization can put next to the

pipeline's build and run costs from Chapter 7's timeline and decide, with an actual number rather than an assumption, whether this use case earned its investment.

Every figure in this example is illustrative, not a claim about what any specific institution should expect — the point is the shape of the calculation, not these particular dollar amounts. An FSI organization running this for real replaces the assumed manual review time, the token pricing, the Fluxnova hosting allocation, and the analyst cost rate with its own numbers, but the structure of the comparison doesn't change.

15.7 Governing Cost the Same Way We Govern Risk

Everything in Chapters 9 and 10 established a pattern: a risk gets a control, a control gets a requirement and a configuration, and the pair lives in the CALM specification where it's reviewed like any other change. Cost governance follows exactly the same pattern rather than needing a separate mechanism — the FinOps discipline of budget alerts, usage caps, and chargeback tagging is just another control, expressed the same way:

```
{
  "nodes": [
    {
      "unique-id": "kyc-agentic-subprocess",
      "node-type": "service",
      "name": "KYC document review (agentic ad-hoc subprocess)",
      "controls": {
        "cost-governance": {
          "description": "Token budget ceiling for this use case, reviewed
  quarterly against Section 15.6's actuals",
          "requirements": [
            {
              "control-requirement-url": "https://fsi.example/controls/schema/
  token-budget-ceiling.json",
              "control-config-url": "https://fsi.example/controls/config/kyc-t
  oken-budget-2026q3.json"
            }
          ]
        }
      }
    },
    {
      "unique-id": "fluxnova-engine",
      "node-type": "system",
      "name": "Fluxnova orchestration engine",
      "controls": {
        "cost-allocation": {
```

```
          "description": "Shared-infrastructure hosting and support cost,
  amortized across every use case this engine runs",
          "requirements": [
            {
              "control-requirement-url": "https://fsi.example/controls/schema/
  shared-infra-allocation.json",
              "control-config-url": "https://fsi.example/controls/config/fluxn
  ova-hosting-allocation-2026q3.json"
            }
          ]
        }
      }
    }
  ]
}
```

Two things are worth noting about this pair of controls. The first, on the agent node, is a hard ceiling — the kind of guardrail that can block a release the same way a failing compliance check does in Chapter 4's CI/CD integration, catching a runaway prompt or an unbounded reasoning loop before it becomes next month's surprise bill. The second, on the Fluxnova engine node itself, isn't a limit so much as an allocation rule — it's what makes Section 15.4's amortization concrete and auditable rather than a one-off spreadsheet calculation, giving even a "free" open-source component an explicit, reviewable cost tag in the same file where its business-logic behavior is governed.

15.8 Building Value Accountability into the Pipeline

This connects back to the pipeline itself, not just to a reporting exercise sitting alongside it. Chapter 9's Step 1 already asks a team to write a use case narrative before anything is built; an FSI organization serious about tokenomics extends that narrative with a stated expected value — what this use case is supposed to save or generate, and roughly how, before a single CALM node exists. That single addition is what turns Section 15.6's report from a retrospective curiosity into an accountability mechanism: the use case's actual net benefit, measured continuously through the same observability layer Chapter 14 already built, can be compared against the value it was approved on the strength of. A use case that's tracking well below its stated expected value is exactly the kind of finding that belongs in Chapter 7's Step 8 feedback loop alongside a drifting escalation rate — not a compliance problem, but a value problem the same continuous-monitoring discipline is well suited to catch.

This is also, ultimately, why tokenomics belongs in a governance guide at

all rather than being left entirely to finance. Chapter 6 argued that AIGF's evidence supports an FSI organization's case for AIUC-1 certification when it's warranted; this chapter's evidence supports the parallel case an institution has to make to itself — not just that a use case is compliant and reliable, but that building it was the right call in the first place, with a number attached rather than an assumption. And as the Tokenomics Foundation's own standards mature, that number stops being an FSI organization's own private calculation and starts being one it can compare against an industry benchmark — the same trajectory AIGF has already run for risk.

15.9 A Control Plane Screen for Tokenomics

Chapter 12's control plane was built to answer one question continuously: does the deployed system still match what was declared. Tokenomics deserves the same continuous treatment rather than living only in the monthly-report format Section 15.6 walked through — a budget that quietly overruns for three weeks before anyone notices it is a governance gap of exactly the same shape as a control silently failing to fire.

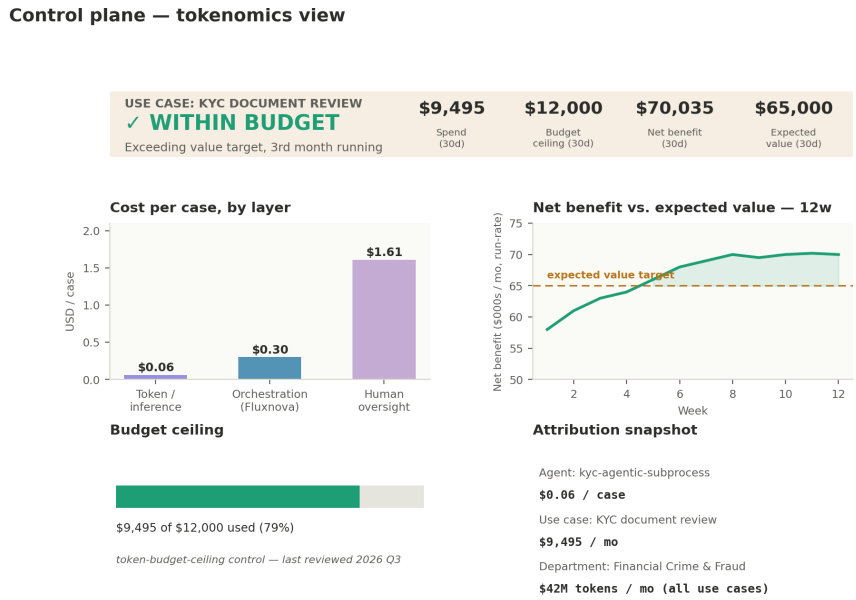


Figure 8. A tokenomics screen in the same control plane: live cost-per-case by layer, net benefit tracked continuously against the expected value stated at Step 1, and a budget ceiling shown as a live position rather than a monthly surprise.

This is the same status-first pattern as Figure 7, applied to cost and value instead of structural conformance. The banner leads with the one judgment that matters — within budget or not — the same way Figure 7 leads with

conformant or not. The budget ceiling panel is Section 15.7's token-budget-ceiling control made visible as a live position rather than something only checked at CI/CD time; the net benefit trend is Section 15.8's accountability argument made continuous, showing the use case tracking above its stated expected value for three consecutive months rather than requiring someone to compile that comparison by hand. And because every figure here is attributed by agent, use case, and department per Section 15.2, the same screen is also where a Governance Layer reviewer would go to answer "why did this month's spend move" without waiting for the next scheduled report.

15.10 One Control Plane, Four Views

Chapters 1, 2, 12, 14, and this chapter have each described a different lens on the same underlying system — the risk catalog, the deployed conformance state, the operational telemetry, and now cost and value. None of them needs to live in a separate tool a Governance Layer reviewer has to remember to check. What ties them together is a single entry point:

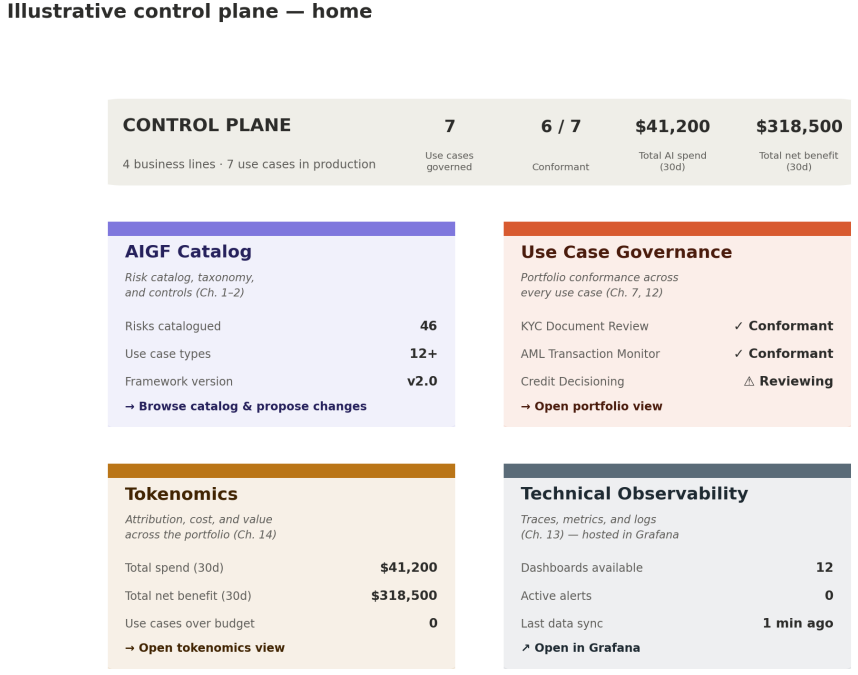


Figure 9. A portfolio-level home screen for the control plane: one launcher into the AIGF catalog, use case governance across the whole portfolio, tokenomics, and technical observability.

The four tiles are deliberately not four different products. The AIGF Cat-

alog tile is a window onto the same risk-and-taxonomy content Chapter 2 described — with a path to propose a change back to it, the same contribution flow Chapter 9 walked through for the original KYC pull request. The Use Case Governance tile is Figure 7's per-use-case conformance screen, rolled up to portfolio level, so a reviewer sees every use case's status at a glance before drilling into one. The Tokenomics tile is Figure 8, similarly rolled up across the whole portfolio rather than scoped to a single use case. And Technical Observability is deliberately an outbound link rather than a fourth native screen — Chapter 14's dashboards, wherever an FSI organization already hosts them (Grafana being the common choice this guide has referenced throughout), stay exactly where they are. The control plane's job is to be the one place that knows those dashboards exist and route a reviewer to them, not to rebuild an operational monitoring tool that already does its job well.

This is also where the argument from Chapter 1 closes all the way around. The pipeline's five stages, the fragmented inputs feeding them, the Controls-stage checks from Chapter 12, and the cost-and-value accounting from this chapter are not four separate initiatives an FSI organization happens to be running at once — they're four views onto one governed system, and a control plane home screen is simply what makes that fact visible to the people who have to act on it.

Conclusion — Why This Matters More Than Another Framework

The real point of Governance-as-Code isn't that AIGF got more detailed — it's that governance stopped being a separate artifact that trails behind engineering and became part of the same system engineers already build, test, and ship through. That's the difference between a risk catalog living in a wiki that nobody opens until an audit, and a risk catalog that shapes what gets built the moment it's written, because the pipeline enforces it automatically. For an FSI organization building its own agentic AI, this reframes the build itself: governance isn't a separate workstream running in parallel to engineering and periodically checking in. It's the substrate the engineering pipeline runs on.

What This Guide Covered

Ten chapters, one continuous argument. Here's the path in one pass:

- **The Introduction** named the pitfalls that show up when FSI organizations build agentic AI without a shared foundation — governance bolted on at the end, duplicated risk assessment, shadow AI, nondeterminism that survives a proof of concept and breaks in production — and made the case for the open-source route over a proprietary black box, ending with why that route gets an FSI organization to ROI faster.
- **Chapter 1** introduced the pipeline itself: fragmented inputs — regulations, guidelines, standards, best practices — synthesized into five stages: policy definition, architecture, controls, orchestration, and observability and feedback, running as a loop rather than a line, with every component named and mapped to its stage.
- **Chapter 2** went stage by stage through what each one actually does, from AIGF's risk catalog and taxonomy through CALM, CI/CD enforcement, and the observability layer that closes the loop.
- **Chapter 3** made the case for Fluxnova specifically: why LLM nondeterminism is a genuine compliance liability, and how a deterministic process backbone around a bounded agent turns "trust the model" into "prove the process."
- **Chapter 4** got concrete for engineers — a hello-world walkthrough of CALM's nodes, relationships, and controls, and a clear line between what agentic design patterns give you (capability) and what they leave out (compliance), which is exactly the gap Governance-as-Code fills.

- **Chapter 5** did the same for GRC teams, then walked the full lifecycle from estimation through production, showing how GRC's involvement shifts from one late, expensive gate to a series of small, well-timed touchpoints.
- **Chapter 6** placed AIGF alongside AIUC-1 — one the internal, continuous engine; the other the external, insured credential — and showed why an FSI organization ends up wanting both, not one instead of the other.
- **Chapter 7** turned the nine-step pipeline into an actual project: who's really doing the work in an FSI organization's hub-and-spoke-and-governance structure, an illustrative timeline, and how to run the build steps as iterative increments instead of a single cascade.
- **Chapter 9** worked a real use case end to end for the first time — an agentic KYC document reviewer — from classification through risk assessment to a merged pull request against the AIGF repository itself.
- **Chapter 10** picked that use case up and turned its classification into an actual solution architecture: a selected reference pattern, and a CALM specification naming not just the business logic but the infrastructure underneath it — the Fluxnova engine, its DMN service, and the LLM provider platform.
- **Chapter 11** finished the build: the KYC process modeled in Fluxnova, an agentic ad-hoc subprocess bounding exactly what the LLM can do, a DMN table that can never auto-clear a sanctions hit, and the concrete argument for why that satisfies the determinism requirement Chapter 3 raised.
- **Chapter 12** turned that design into a passing build: automated CI/CD checks confirming the CALM file's structure, its control coverage against Chapter 9's risk shortlist, and the DMN table's behavior across every input combination that matters — the gate between building the KYC process and trusting it.
- **Chapter 13** filled in a term this guide had used since Chapter 1 without ever defining it: evals. Eval-driven development, curated goldens, and a worked failure in the KYC extraction tool showed how to measure the quality and safety of what happens inside the boundary Fluxnova already draws — the question determinism alone can't answer.
- **Chapter 14** closed the loop into production: how traces, metrics, and logs turn Chapter 11's process into task-level execution data, how that same data becomes auditable evidence that specific controls held,

and how a drifting trend on a dashboard becomes the trigger for Chapter 7's Step 8 feedback loop.

- **Chapter 15** extended that same data one step further, into cost and value: attributing token consumption by agent, use case, and department, and setting that cost against the value a use case was built to create — so an FSI organization can answer not just "is this compliant" but "was building this the right call."

The Benefits, Taken Together

Individually, each chapter argued for one piece of this. Put together, adopting the Governance-as-Code pipeline changes what an institution can do, not just how it does it:

- **Faster time to production.** Compliance checks run automatically, in the pipeline, rather than as a manual gate at the end — Chapter 4's point about build-time signals and Chapter 5's estimation-to-production walkthrough both land on the same conclusion: less rework, fewer late surprises, a shorter path from idea to running system.
- **Reusable infrastructure instead of one-off effort.** The taxonomy, the risk catalog, the reference architectures, and the CI/CD checks all persist. A FSI organization's second use case moves faster than its first, and — because AIGF is open source — every FSI organization's contribution shortens the path for every other institution using the framework, as Chapter 9's pull-request walkthrough showed directly.
- **Audit and certification evidence as a byproduct, not a project.** Because execution is traced automatically (Chapter 11) and every control traces back to a specific, approved line in a risk assessment (Chapter 10), both an internal audit and an external AIUC-1 evaluation (Chapter 6) start from evidence that already exists rather than a costly evidence-gathering exercise built from scratch.
- **A genuine answer to "how do you know it's compliant," not an assurance.** The determinism argument running from Chapter 3 through Chapter 11 means an FSI organization can point to a specific gateway, a specific DMN row, a specific control — not a claim that the model behaves well.
- **Organizational clarity.** Chapter 7's hub-and-spoke-and-governance model gives every one of the tasks in this guide a named owner, so governance is neither a bottleneck at the Hub nor an inconsistent free-for-all across the Spokes.
- **Resilience to what you don't know yet.** Chapter 7's iterative delivery

model and Chapter 1's continuous feedback loop mean the system is built to absorb what production teaches it — an unanticipated escalation pattern updates the risk catalog and the DMN table, rather than waiting for the next major release.

- **A stronger position with regulators, and with the industry.** Because AIGF's risk catalog already maps to frameworks like the EU AI Act, FCA Consumer Duty, and PRA operational resilience requirements, and because it's shaped by the same institutions that answer to the same regulators, an FSI organization using it is speaking a language regulators and peers already recognize, not defending a bespoke internal framework nobody else has seen.

None of this requires waiting for a perfect, fully-built pipeline before starting. The KYC use case in Chapters 9 through 11 shows the whole path end to end, but Chapter 7's iterative model is the honest way to actually begin: one behavioral slice, one small loop through classification, architecture, and CALM, shipped into a production loop that never resets. The pipeline gets more valuable with every use case that runs through it — which is exactly the argument for starting now rather than waiting for the tenth use case to make the first one look easy.

References

This guide draws on the following public sources. Where a fact, quote, or figure in this guide is attributed to FINOS, Fluxnova, CALM, or AIUC-1, it traces back to one of these.

FINOS AI Governance Framework (AIGF)

AI Governance Framework — GitHub repository — https://github.com/finos/ai-governance-framework

AI Governance Framework — published site (risk catalog, taxonomy, heuristic assessment) — https://air-governance-framework.finos.org/

AI Governance Framework — governance policy (Participants, Maintainers, consensus process) — https://github.com/finos/ai-governance-framework/blob/main/GOVERNANCE.md

FINOS AI Governance Framework v2.0 announcement — https://www.finos.org/blog/ai-governance-framework-v2.0-addressing-agentic-ai-risks-in-a-rapidly-evolving-landscape

AI Readiness SIG — GitHub repository (origin of AIGF) — https://github.com/finos/ai readiness

Governance-as-Code, the FINOS AI Fund, and Common Cloud Controls

Governance as Code — AI at FINOS — https://ai.finos.org/governance-as-code/

AI at FINOS — program homepage — https://ai.finos.org/

FINOS Launches AI Fund — press release — https://www.finos.org/press/finos-launches-ai-fund

iTmethods joins FINOS and the Agentic AI Foundation — Governance-as-Code pipeline context — https://www.prnewswire.com/news-releases/itmethods-joins-the-linux-foundation-finos-and-agentic-ai-foundation-to-advance-governance-standards-for-regulated-agentic-ai-302816923.html

CALM (Common Architecture Language Model)

CALM — project homepage and guided journeys — https://calm.finos.org/

CALM — "What is CALM?" introduction — https://calm.finos.org/introduction/what-is-calm/

architecture-as-code — GitHub monorepo (CALM specification, CLI, Calm Hub, VS Code plugin) — https://github.com/finos/architecture-as-code

architecture-as-code — CALM specification source — https://github.com/finos/archite as-code/tree/main/calm

calm-cli — npm package documentation — https://www.npmjs.com/package/@finos/ cli

Fluxnova

Auditable Agentic Orchestration: From Autonomous Systems to Governed Execution — Scott Logic (introduces the Agentic Subprocess) — https://blog.scottlogic.com/2026/07/08/auditable-agentic-orchestration.html

fluxnova-bpm-platform — GitHub repository — https://github.com/finos/fluxnova-bpm-platform

fluxnova-modeler — GitHub repository — https://github.com/finos/fluxnova-modeler

fluxnova-modeler — releases — https://github.com/finos/fluxnova-modeler/releases

Fluxnova 1.0 Released — announcement discussion — https://github.com/finos/fluxnc bpm-platform/discussions/6

Fluxnova 2.0 & 3.0 release roadmap — discussion (restricted variables, MCP integration) — https://github.com/finos/fluxnova-bpm-platform/discussions/59

AIUC-1

AIUC — standard and insurance homepage — https://aiuc.com/

AIUC-1 — standard homepage — https://www.aiuc-1.com/

What Is AIUC-1? The First Security Standard Built for AI Agents — Workstreet — https://www.workstreet.com/blog/what-is-aiuc-1

AIUC-1 AI Agent Certification: The Complete Guide — https://axipro.co/aiuc-1-certification/

Rajiv Dattani is bringing insurance to the AI agent boom — Fast Company — https://www.fastcompany.com/91550776/rajiv-dattani-is-bringing-insurance-to-the-ai-agent-boom

Tokenomics

Linux Foundation Launches the Tokenomics Foundation to Define the Economics and ROI of AI Value — https://www.linuxfoundation.org/press/linux-foundation-launches-the-tokenomics-foundation-to-define-the-economics-and-roi-of-ai-value

Tokenomics Foundation — homepage — https://www.tokeneconomics.com/

Big-T Notation project — Tokenomics Foundation — https://www.tokeneconomics.com
t-notation/

FOCUS — FinOps Open Cost and Usage Specification — https://focus.finops.org/

The pivot to tokenomics: Navigating AI's new spend dynamics — Deloitte
— https://www.deloitte.com/content/dam/assets-zone3/us/en/docs/services/consult
to-navigate-economics-of-ai.pdf

Evals

Eval Driven Development: What it is, how to do it right, and real examples to learn from — DeepEval — https://deepeval.com/blog/eval-driven-development

DeepEval — GitHub repository (open-source LLM evaluation framework)
— https://github.com/confident-ai/deepeval

Eval harness: What it is, how to use it, and why you should care — DeepEval — https://deepeval.com/blog/what-is-an-eval-harness

AI Agent Evaluation guide — DeepEval — https://deepeval.com/guides/guides-ai-agent-evaluation