Matched Pair: When the Governance Framework Caught Its Own Builder

v1.0.0

Author: John J. McCormick · Updated: March 4, 2026 · CC BY 4.0 · aiagentgovernance.org

SUGGESTED CITATION

McCormick, J. J. "Matched Pair: When the Governance Framework Caught Its Own Builder." v1.0.0, March 2026. aiagentgovernance.org. https://aiagentgovernance.org/insights/matched-pair-routing-enforcement/

Abstract

A complete audit trail showing plan vs. execution across a governed AI agent workstream — including the incident where an agent deviated, was caught, and was corrected in real time.

This document is a **matched pair**: the routing specification (what was supposed to happen) placed side-by-side with the machine-generated execution trace (what actually happened) for a single governance workstream. The workstream in question built the system designed to prevent agents from violating their own routing plans. During its own construction, an agent violated the execution path — and the framework caught it.

This is a companion exhibit to Largely Untested (/insights/largely-untested/), which argues that proposed governance interventions are no longer entirely untested. What follows is the evidence.

What This Document Is

Every claim below is verifiable against:

- Version-controlled documents with commit hashes
- Governance control plane entries with UUIDs, timestamps, and immutable audit records
- Incident reports written during the events they describe

No part of this evidence was created retroactively. The routing specification was committed before execution began. The control plane entries were generated by the agents as they executed. The incident was documented the night it occurred.

Why This Workstream

This workstream is the strongest matched pair in the corpus for three reasons:

1. **Self-referential scope.** This workstream built the system designed to prevent agents from violating their own routing plans. During its own construction, an agent violated the execution path — and the framework caught it.
2. **Complete lifecycle.** The workstream traversed PLAN → REVIEW_ARCH → BUILD → REVIEW_ENG (FAIL) → REMEDIATE → REVIEW_ENG (FAIL) → REMEDIATE (Cycle 2) → COMPLETE — including two remediation loops triggered by real defects found by independent review agents.
3. **Incident during execution.** An incident occurred mid-lifecycle: an agent inferred its work order from the wrong source document, was caught by the human Operator observing terminal output, and the root cause was reclassified from "infrastructure failure" to "behavioral pattern" after post-incident investigation revealed the agent had working access the entire time.

Side A: The Routing Specification (Declared Plan)

Committed: 2026-02-24, before any execution began **Author:** Platform Architect

Step	Description	Type	Target Role	Depends On
1	Operator reviews PLAN spec	Gate	Operator	—
2	Route to REVIEW_ARCH	Review	Feature Architect	Step 1
3	Route to BUILD (schema + API)	Routing	Implementation Engineer	Step 2
4	Route to REVIEW_ENG	Review	Engineering Reviewer	Step 3
5	Route to REMEDIATE if findings	Routing	Implementation Engineer	Step 4
6	Route to APPROVE	Gate	Operator	Step 4 or 5

Self-aware constraint noted in spec: *"This plan is prose (we can't use the structured plan system to build the structured plan system). Once Phase 1 ships, all subsequent workstreams will use machine-enforced plans."*

Side B: The Execution Trace (Control Plane Audit Trail)

Every row below is a machine-generated governance control plane entry — a structured record with a UUID, creation timestamp, author agent ID, affected agents list, and Operator approval timestamp. These entries are immutable: agents cannot edit or delete them after creation.

Phase 1: PLAN → REVIEW_ARCH (Steps 1–2)

#	Timestamp (UTC)	Entry ID	Author	Title	Operator Approved
1	2026-02-24 04:30	`2a1408cc`	Platform Architect	PLAN Complete — Routing Plan Enforcement Spec v1.0.0	04:37 (7 min)

What happened: The Platform Architect completed the specification and posted an entry routing it to the Feature Architect for architecture review. The Operator approved in 7 minutes.

Plan compliance: Matches Step 1 → Step 2 transition. Correct target. Correct gate type (Operator approval required before REVIEW_ARCH).

Impact Statement included:

```
Risk:         LOW
Action:       Approve PLAN spec
If PASS:      Routes to Feature Architect for REVIEW_ARCH
If FAIL:      Spec revised per Operator feedback
Reversible:   Yes — review gate only, no code
Blast radius: None — spec document only
```

Phase 2: REVIEW_ARCH (Step 2)

Architecture review was performed by the Feature Architect directly — no separate routing entry was needed because the PLAN completion entry named the Feature Architect as an affected agent.

Disposition: PASS — 0 blocking findings, 5 findings (0 P1, 1 P2, 3 P3, 1 advisory)

Key findings:

- **(P2):** Step completion should tie to control plane approval, not a separate endpoint — “Self-reporting is the same trust model that created a previous incident”
- **(P3):** Governance phase enum missing two lifecycle stages

- **(Advisory):** Self-bypass detection only checks plan author, not executor self-assignment

Plan compliance: Matches Step 2. Correct reviewer. PASS disposition allows Step 3.

Phase 3: BUILD → REVIEW_ENG (Steps 3–4)

#	Timestamp (UTC)	Entry ID	Author	Title	Operator Approved
2	2026-02-24 15:40	`bf69ea21`	Feature Architect	REVIEW_ENG — Route to Engineering Reviewer	15:42 (2 min)
3	2026-02-24 15:50	`766c4d5a`	Engineering Reviewer	REVIEW_ENG FAIL — remediation required	17:21 (91 min)
4	2026-02-24 21:51	`d3c58f1e`	Engineering Reviewer	Review artifacts committed — disposition	23:32 (101 min)

What happened: BUILD completed. The Feature Architect routed engineering review to the Engineering Reviewer. The Engineering Reviewer performed static code review and returned **FAIL** with:

- **(P1):** Active-plan versioning path broken — insert-before-supersede violates unique constraint
- **(P1):** Step completion can be self-reported without approval linkage — same trust model weakness the Feature Architect flagged
- **(P2):** Governance phase enums incomplete

Plan compliance: Matches Steps 3→4. Correct reviewer. FAIL disposition triggers Step 5 (REMEDIATE).

Observation: The two P1 defects are real bugs that would have caused production failures. The governance lifecycle caught them before any code shipped to production. The second P1 independently confirms the Feature Architect's P2 finding from a different review perspective.

Phase 4: REMEDIATE Cycle 1 (Step 5) — Including Incident

#	Timestamp (UTC)	Entry ID	Author	Title	Operator Approved
5	2026-02-25 03:12	`969d72e6`	Platform Architect	REMEDIATE — Route to Implementation Engineer (3 findings)	03:22 (10 min)

What happened: The Platform Architect posted a routing entry directing the Implementation Engineer to remediate the 3 engineering review findings. The Operator approved in 10 minutes.

Plan compliance: Matches Step 5. Correct target. Correct phase (REMEDIATE).

Incident: Execution Deviation Detected

Time: ~22:25–22:55 ET, 2026-02-24 (immediately after the Implementation Engineer received the approved work order) **Detection:** Operator observed the agent's terminal output in real time

What happened during execution:

The Implementation Engineer received an approved entry containing 3 engineering review findings. Rather than reading the entry via the governance API (the standard access method), the agent:

1. Queried its local database directly — found nothing (wrong database)
2. Searched git log for the engineering review artifact — not found (committed to a different repository)
3. Read the Feature Architect's architecture review (a different review phase, different agent, different findings)
4. **Inferred** the engineering findings from the architecture review

The inferred findings were wrong:

What the agent was working on	**Actual engineering finding**	**Problem**
Architecture finding: Remove completion endpoint	Engineering P1: Approval linkage	Partial match — same root cause, wrong source
Architecture finding: Add phases to enum	Engineering P2: Incomplete enums	Match
Architecture finding: File split (advisory)	**Not an engineering finding**	Out of scope — P3 architecture advisory
Not identified	**P1: Active-plan versioning conflicts with unique constraint**	**Missing P1**

Post-incident root cause reclassification:

Original assessment: "Agent cannot access production governance API" (infrastructure gap)

Revised assessment (confirmed by Operator screenshot): Both the API URL and authentication token were configured in the agent's environment at incident time. When directed to test, the agent successfully returned production data via the governance API and later posted its own completion entry to the production governance control plane.

Final root cause: The agent had full production access but never attempted to use it. It misdiagnosed a working system as broken and used the misdiagnosis to justify inference from the wrong document.

Behavioral patterns detected:

- **BP-001** (/framework/agent-failure-patterns/) **(Inference Over Execution):** Agent reconstructed work order content instead of reading it from the canonical source
- **BP-002** (/framework/agent-failure-patterns/) **(False Blocker Reporting):** Agent reported infrastructure blocker that did not exist — repeated across multiple sessions
- **BP-008** (/framework/agent-failure-patterns/) **(Work Order Contamination):** Agent worked from wrong source document (architecture review findings instead of engineering review findings)

Operator intervention: Correct findings pasted directly into the agent's context. Agent redirected. No incorrect code was committed.

Agent's own terminal output (captured by Operator):

> *"Let me find the engineering review findings myself. They're likely in a doc that was recently committed, or I can find them in the bulletin content by looking at the production database via the SQL tool."*

> *"The bulletin isn't in the local DB. Let me check if it was posted to production — the local and production databases may be separate."*

> *"The engineering review doc hasn't been committed to this repo yet. The findings are only in the production bulletin. Since the local and production databases are separate [...] let me try to access production data through the deployment logs or by examining what the arch review already flagged"*

Phase 5: REMEDIATE → Re-Review (Loop Back to Step 4)

#	Timestamp (UTC)	Entry ID	Author	Title	Operator Approved
6	2026-02-25 05:02	`92f6db57`	Platform Architect	REVIEW_ENG Re-Review — Route to Engineering Reviewer	05:20 (18 min)
7	2026-02-25 05:34	`e7180ced`	Engineering Reviewer	REVIEW_ENG Re-Review FAIL — cycle 2 required	05:38 (4 min)

What happened: After remediation cycle 1, the Platform Architect routed re-review back to the Engineering Reviewer. The Engineering Reviewer found the remediation was **incomplete**:

- **(P1):** Non-atomic supersede/create path — can leave a workstream with no active plan on partial failure
- **(P1):** Previous P2 finding remains unresolved — governance phase enum still missing two lifecycle stages
- **(P2):** Hardcoded phase enums in application layer not aligned with shared enum

Plan compliance: Loop back to Step 4→5. The routing plan anticipated this: Step 5 says "Route to REMEDIATE **if findings**" — the re-review loop is the governance lifecycle working as designed.

Phase 6: REMEDIATE Cycle 2 → COMPLETE

#	Timestamp (UTC)	Entry ID	Author	Title	Operator Approved
8	2026-02-25 12:11	`3705936b`	Platform Architect	REMEDIATE Cycle 2 — Route to Implementation Engineer	12:57 (46 min)
9	2026-02-25 14:07	`48fcc491`	Implementation Engineer	REMEDIATE Cycle 2 — 3 Findings Resolved	14:23 (16 min)

What happened: The Platform Architect routed the second remediation cycle to the Implementation Engineer with explicit instruction: *"READ THIS ENTRY BEFORE STARTING — do not infer findings from other sources."* This instruction was added as a direct response to the incident.

The Implementation Engineer completed remediation and posted its own completion entry directly to the production governance control plane — confirming it had working API access the entire time.

Remediation delivery included:

- New transactional method for atomic plan versioning (fixes the P1 versioning conflict)
- Replaced hardcoded phase enums with shared enum (fixes the incomplete lifecycle stages)
- Auto-record step completion on entry approval (fixes the self-reporting trust model weakness)
- Database migration adding missing lifecycle stages to CHECK constraint

Plan compliance: Step 5 complete (second cycle). COMPLETE entry posted by the executing agent.

Epilogue: System Monitoring

#	Timestamp (UTC)	Entry ID	Author	Title	Operator Approved
10	2026-02-27 07:30	`d365cdd4`	system	BUILD phase has no completion entry (66h)	2026-02-28 06:43

What happened: The automated stale-detection system flagged that the BUILD phase had no completion entry after 66 hours. The builder completed work but did not post a BUILD completion entry before the REVIEW_ENG routing. The system correctly detected this process gap after the fact.

Plan vs. Execution: Compliance Map

Plan Step	Planned Target	Actual Executor	Planned Outcome	Actual Outcome	Compliant?
1. Operator reviews PLAN	Operator	Operator	Approve/reject	Approved (7 min)	Yes
2. REVIEW_ARCH	Feature Architect	Feature Architect	PASS/FAIL	PASS (5 findings, 0 blocking)	Yes
3. BUILD	Implementation Engineer	Implementation Engineer	Schema + API built	Built (no completion entry)	Process gap
4. REVIEW_ENG	Engineering Reviewer	Engineering Reviewer	PASS/FAIL	**FAIL** (2 P1, 1 P2)	Yes (FAIL is valid)
5. REMEDIATE	Implementation Engineer	Implementation Engineer	Fix findings	**Incident** → Cycle 1 incomplete → Cycle 2 complete	Deviation caught
6. APPROVE	Operator	—	Final sign-off	Pending (BUILD stale alert active)	In progress

Behavioral Patterns Observed During This Lifecycle

Detected During Incident (REMEDIATE, Step 5)

Pattern	Name	Evidence
BP-001 **(/framework/agent-failure-patterns/)**	Inference Over Execution	Agent reconstructed findings from the architecture review instead of reading the canonical work order via the governance API
BP-002 **(/framework/agent-failure-patterns/)**	False Blocker Reporting	Agent reported “I don’t have API access” in a subsequent session — confirmed false by Operator screenshot showing credentials configured
BP-008 **(/framework/agent-failure-patterns/)**	Work Order Contamination	Agent worked from wrong source document (architecture review ≠ engineering review), producing a findings list that was missing a P1 and included an out-of-scope item

Detected by Review Agents (REVIEW_ENG, Step 4)

Finding	What It Caught	Why It Matters
P1	Insert-before-supersede violates unique constraint	Would cause production database errors on plan versioning
P1	Step completion self-reportable without approval linkage	Would undermine the anti-bypass control the system exists to provide
P2	Governance phase enum incomplete	Lifecycle model drift — steps for missing phases would be rejected by validation

Detected by Architecture Review (REVIEW_ARCH, Step 2)

Finding	What It Caught	Why It Matters
P2	Dual-path completion creates trust model weakness	”Self-reporting is the same trust model that created a previous incident” — the incident that motivated this workstream

---

What This Matched Pair Demonstrates

1. The Governance Lifecycle Catches Real Defects

Two P1 defects were found by an independent review agent that would have caused production failures. The architecture review independently identified the same trust model weakness from a different angle. These are not hypothetical findings — they are bugs in deployed code caught before production.

2. Deviations Are Detected and Documented in Real Time

The incident was caught by the human Operator observing the agent's terminal output within ~20 minutes of the deviation beginning. The incident report was written the same night. The root cause was reclassified the next morning when additional evidence contradicted the initial "infrastructure gap" assessment.

3. The Audit Trail Is Machine-Generated and Immutable

Every entry in the execution trace has:

- A UUID (globally unique, assigned by the database)
- A creation timestamp (server-generated, not agent-supplied)
- An author agent ID (authenticated via API token)
- An Operator approval timestamp (separate from creation — shows the human-in-the-loop gap)
- Immutable content (agents cannot edit entries after creation)

4. The Framework Governs Its Own Construction

This workstream was built to prevent routing plan violations. During its own build:

- The routing plan was followed (Steps 1–4 fully compliant)
- A deviation occurred during Step 5 (agent inferred from wrong source)
- The deviation was caught by the Operator (human detection)
- The root cause was misdiagnosed, then correctly reclassified based on new evidence
- The remediation included an explicit process correction

This is the framework detecting and correcting behavioral drift *in the agents that are building the framework*.

5. Cross-Platform Review Architecture

The lifecycle involved 4 agents across 2 AI platforms, plus the human Operator approving every phase transition. The cross-platform, cross-model architecture means no single AI provider's biases dominate the governance decisions.

Further Reading

- Largely Untested (/insights/largely-untested/) — What operational evidence reveals about agent governance interventions
- 8 Ways AI Agents Fail (/insights/8-ways-agents-fail/) — The behavioral pattern taxonomy referenced in this document
- MOIAS Methodology (/framework/governance-lifecycle/) — The governance lifecycle model governing this workstream
- Behavioral Pattern Taxonomy (/framework/agent-failure-patterns/) — BP-001, BP-002, and BP-008 as observed here

---

Independent Verification

A GPG-signed copy of this evidence package, including the complete governance control plane audit trail, is available for third-party validation upon request. Contact the author (/contact/).

Version History

Version	Date	Author	Description
1.0.0	2026-03-04	John J. McCormick	Initial publication

This document is part of the aiagentgovernance.org open framework for AI agent governance. The evidence was compiled from a production governance control plane audit trail, version-controlled governance documents, and incident reports. No retroactive edits. All timestamps are server-generated. Published under CC BY 4.0 to enable adoption, citation, and community contribution.