

Where AI Governance Breaks Between Approval and Action

By Stephan Pochet

AI governance depends on controls that remain effective from the source information through approval and execution. This essay examines human review, data ownership, system changes, permissions, and incident response.

An assistant prepares a payment to a supplier. Someone checks the invoice and approves it. Before the payment goes through, the supplier's bank details change.

Does the approval still hold?

That question belongs in the design of the workflow. If it first comes up during an investigation, the organization is already trying to reconstruct a decision it should have controlled.

AI governance often becomes tangible at moments like this. A policy names an owner. An assessment describes the risks. But an ordinary change between review and execution can expose a gap that neither document resolves.

Five areas of work help close that gap: human review, data governance, lifecycle records, enforceable permissions, and monitoring. They are familiar subjects. The difficult part is making them work across the same transaction.

Start with the action

The supplier-payment example is hypothetical, but it gives a governance team a useful place to begin. It has a recognizable purpose and a result that matters outside the application. Money either reaches the intended account or it does not. The organization has to account for that outcome even if several systems contributed to it.

Start by drawing the boundary of the assistant's work. Preparing a recommendation is one activity. Submitting instructions to a payment service is another. A team that calls both activities payment assistance can miss the moment when the application acquires authority to affect someone else.

This distinction also changes the evidence needed for approval. A useful drafting assistant may tolerate errors that a reviewer can correct before anything happens. A system allowed to execute needs controls over the action itself. An accurate answer in a test set does not establish that the system will respect a user's permissions when connected to a live account.

Follow the work across organizational boundaries as well. The application team may maintain the assistant while finance controls supplier records. Another team may own

identity management. Each can perform its assigned work correctly while leaving a gap between services. The purpose of the review is to locate those gaps while they can still be addressed deliberately.

An inventory becomes useful when it tells you who can answer the next question. If a record merely names a department, ask who in that department can authorize a change or stop an operation. Someone needs to be reachable when the process encounters a condition the design did not anticipate.

Approval needs a clear boundary

In the payment example, the reviewer needs to know precisely what is being approved: the supplier, the amount, the destination, and the evidence supporting the payment. The system must establish which changes invalidate that approval.

Otherwise, a person can approve one action and appear responsible for another.

The reviewer also needs enough time and information to question the recommendation. A queue full of urgent requests can turn review into a habit of clicking through. Teams should test whether people can spot a wrong amount or an unsupported supplier change under realistic working conditions.

Then test a refusal. Check that execution actually stops, including work already queued in a connected system. A recorded rejection is of little use if the payment still goes through.

The cost of asking a person

Human review has a cost, and a team should acknowledge it. Every approval request consumes attention. If the application asks for confirmation on routine, easily reversible steps, reviewers may become less attentive when the consequential request arrives. The design should distinguish those situations according to the risk of the action.

Consider what the reviewer can independently establish. Showing an AI-generated explanation beside an AI-generated recommendation may create the appearance of corroboration while both depend on the same faulty source. Provide access to the underlying record and make unresolved discrepancies visible. The person should not have to discover that the supporting material was incomplete after approving the transaction.

Responsibility also needs a practical limit. A reviewer cannot meaningfully accept every possible consequence of a system they cannot inspect. Define the decision they own and the checks performed elsewhere. That makes the approval useful to the person giving it and to anyone examining the record later.

When the reviewer is unavailable, a consequential request should follow a defined fallback. It might remain pending or go to an authorized alternate. The choice depends on the workflow, but silence should not quietly turn into consent. The organization needs to decide what delay it can tolerate before pressure for speed makes that decision informally.

The source matters as much as the answer

An assistant may find a supplier's details in an invoice or pull them from a shared document. Those sources do not necessarily have equal authority.

Someone must decide which record governs the payment and who may change it. That is a data governance decision with an immediate operational consequence. If an invoice conflicts with the approved supplier record, the workflow needs a defined way to resolve the discrepancy.

The same problem appears in less obviously consequential applications. An internal assistant can give a confident answer from a handbook that was replaced last month. Removing the old file from a shared folder may leave a copy in the retrieval index. The correction is complete only when the team checks that the application has stopped using it.

Ownership, access, currency, and correction procedures all matter here. They determine whether the information reaching the reviewer deserves to be relied on.

Correction has to reach the application

Data ownership includes the ability to put something right. Suppose finance confirms that a supplier record is wrong. The team needs to find where the application obtained it and whether other pending recommendations depend on the same information. Correcting the authoritative record is the beginning of that work.

Copies complicate the response. A cache can continue serving an earlier value. A retrieval index may refresh on a schedule. A pending approval may contain a recommendation generated before the correction. The team should decide which of those items must be invalidated and how to verify that the correction reached them.

Access creates a similar problem. An employee may lose permission to view a document while an assistant's search index still exposes its contents. Checking access only when the employee opens the chat does not establish whether each retrieved item is permitted. Test with accounts that have different responsibilities and include a change of role in the test.

Keep the investigation proportionate. A team does not need to collect every available piece of information about a user to establish which document supported a recommendation. Capture what is needed to understand and correct the decision, with restrictions appropriate to the material. An evidence trail that unnecessarily spreads confidential information creates another problem to govern.

Approval belongs to a particular system

A model name tells an auditor only part of the story. The application's instructions, retrieval sources, connected tools, and permissions also affect what it can do.

Suppose an assistant was approved to prepare payment instructions. A later release adds a tool that can submit them. The model may be unchanged, but the application now has

different authority. That change needs to reach whoever is responsible for assessing and approving it.

An inventory identifies the application and its owner. A registry helps track model versions. Deployment records connect those details to the configuration actually running. Together, they should allow the team to establish what was approved and whether the system stayed within those conditions.

There are limits. A third-party provider may not expose enough version information to reconstruct every aspect of a past result. Record that limitation. Do not promise an audit trail the service cannot provide.

Decide which changes reopen the review

A release process needs criteria for returning to governance review. Replacing a foundation model is an obvious candidate. Expanding the population of users can matter just as much. So can connecting a new document repository or allowing the application to operate without a person present.

These changes should be evaluated for what they permit the system to do and whom they affect. A minor software release can introduce a major change in authority. Conversely, a maintenance update may leave the approved use and control boundaries intact. Treating every change as identical can overwhelm reviewers without improving their attention to the changes that matter.

Record the decision and its basis. A later investigation should be able to distinguish a change that was assessed and accepted from one that simply escaped notice. This is also useful for the next team that inherits the application. They need to understand why a restriction exists before deciding whether to remove it.

Rollback deserves its own rehearsal. Restoring an earlier model does not necessarily restore an earlier application if the supplier database, permissions, or connected service has changed. The recovery plan should identify what can be restored safely and what requires a manual process. A rollback button is a promise that needs testing.

Enforce the rule where it can be broken

If the assistant must never change a supplier's bank account, restrict that action in the connected system. A sentence in its instructions is not a sufficient authorization control.

OWASP's guidance on excessive agency makes this practical: limit the available functionality and permissions, and enforce authorization in downstream systems. Human approval is another control, particularly for high-impact actions. Each serves a different purpose. [1]

For the payment workflow, test the routes around the intended process. Can a different tool perform the same prohibited change? Does an administrator's account bypass the restriction? What happens if the approval service is unavailable?

Exceptions need attention too. Someone should own each temporary permission and document why it was granted. Set an expiry when granting it. Otherwise, the next review may discover that an emergency workaround has become the operating model.

Exceptions reveal the real operating model

Imagine that a legitimate payment is repeatedly blocked. Finance needs to complete it, the application team is under pressure, and an administrator can grant broader access in minutes. The technical shortcut is easy. The governance question is what happens to that access once the immediate problem has passed.

An exception record should let the next reviewer understand its scope and the condition for ending it. The person granting it should also know which protections still apply. If a restriction is removed entirely, calling the change temporary does not reduce the exposure while it remains in place.

Review the pattern of exceptions, too. Repeated requests for the same workaround may indicate that the approved workflow does not meet a legitimate business need. The answer may be to redesign the process and assess it properly. Leaving users dependent on recurring overrides makes the formal policy less useful as a description of how work actually happens.

Some exceptions should be refused. If the team cannot establish who will receive a payment or cannot enforce the required authorization, completing the transaction through the assistant may be inappropriate. A manual route can preserve the business function while the technical problem is addressed. That route must still have an accountable owner.

Give an alert somewhere useful to go

Monitoring should help someone make a decision. An alert about an unauthorized tool attempt needs an owner who can investigate it and, when necessary, restrict or stop the workflow.

Logs and traces can help establish which tools were called, what they returned, and which configuration was operating. They do not automatically reveal the model's internal reasoning. They can also contain confidential material, so the evidence needs its own access and retention controls.

The existing AI observability guide provides a starting point for this part of the work. The operational test is straightforward: introduce a controlled failure and follow it through detection, escalation, containment, and recovery. Include the person covering an absence or an out-of-hours incident.

NIST's AI Risk Management Framework includes ongoing monitoring, response, and recovery in its MANAGE function. Those outcomes need to become responsibilities that people can carry out. [2]

Decide what a signal can actually prove

A monitoring measure needs an interpretation. An increase in rejected tool calls might indicate attempted misuse. It might also follow a legitimate change that the permission rules did not accommodate. The signal gives the team a reason to investigate; it does not decide the explanation.

A quiet dashboard can be misleading for a different reason. The application may have stopped sending events. Check the monitoring path itself so the absence of an alert is not mistaken for evidence that everything worked. Where possible, compare the assistant's records with the connected system's record of the action.

The response needs to account for completed work. Stopping the assistant prevents some future activity, but it does not undo an external message or recover a payment already released. An incident process should distinguish containment from correction and identify who can assess the people or records affected.

This is where governance becomes an ongoing responsibility. The team learns something about a failure and has to decide which part of the process changes as a result. That may be a permission boundary, the evidence shown to a reviewer, or the conditions that reopen approval. Closing the incident without considering those changes leaves the same route available for the next failure.

Make the evidence usable by someone else

A governance record should make sense to a person who was not in the room when the decision was made. That is a demanding test. Teams often rely on shared understanding that never reaches the record: a limitation mentioned in a meeting, an assumption about which users would have access, or a promise that a feature would remain disabled.

Write down the conditions that matter to the approval. If the assistant is permitted to draft but cannot execute, say so in a way that the deployment team can verify. Identify the evidence supporting that boundary and what would cause it to be reconsidered. The record should help someone ask a precise question about the live system.

Avoid turning the file into an archive of everything the project produced. A large folder can hide the absence of a clear decision. Organize the evidence around the approved use and make its limitations easy to find. Keep the underlying test results available, with enough context to understand what was tested and what was outside scope.

The same discipline helps when staff change. A new owner should be able to discover why a control exists without rebuilding the project's history from messages. They should also be able to identify an assumption that no longer holds. Governance becomes fragile when maintaining a system depends on remembering who happened to know the answer last year.

Follow one decision all the way through

A useful review can begin with a single transaction. Take it from the source document to the recommendation, the approval, the execution, and the resulting record. Change one material detail along the way. Make a component unavailable. Ask the reviewer to refuse it.

This exercise often reveals questions that separate control reviews miss: who resolves conflicting records, which changes require another approval, and who has the authority to intervene when an alert arrives.

The payment example is illustrative. The same review can be applied to an assistant that closes a case, changes a customer record, or sends an external message, with controls proportionate to the consequences.

The organization should be able to explain how the action happened and why it was permitted. More importantly, the people running the process should be able to prevent an action they have decided must not happen.

Sources

[1] OWASP: Excessive Agency.

[2] NIST AI Risk Management Framework Core.

The scenarios and review practices are editorial implementation examples.

Questions and answers

What makes human approval effective in AI governance?

The reviewer needs a specific action to approve, reliable supporting information, enough time, and authority to refuse it. Material changes must trigger reassessment, and rejection must prevent execution.

What is the difference between an AI inventory and a model registry?

An AI inventory identifies applications, purposes, and owners. A model registry tracks model versions. Deployment records connect those records to the configuration actually operating.

Where should an AI application enforce permissions?

The connected system should enforce authorization for the requested action. Instructions to the model alone are not a sufficient permission boundary.

Do logs explain an AI model's internal reasoning?

No. Logs and traces can record observable actions and configuration details, but they do not automatically reveal internal reasoning. Evidence also needs access and retention controls.