

Assuring agents through change

Connecting release evidence, operational signals and controlled adaptation over an agent's working life.

A release decision has a shelf life. Identify which changes invalidate its assumptions, choose proportionate checks, and connect production signals to decisions about continuing, constraining, correcting or withdrawing an agent workflow.

FOR Engineering leaders, platform operators and assurance teams

Softlander Consultancy FZE LLC | 2026-09-16 | v1.0

The change model, worksheet and scenario are illustrative. No production performance, assurance guarantee or client outcome is claimed.

In this paper

Treat release evidence as conditional	2
Establish the baseline you need to compare	3
Classify changes by what they can invalidate	4
Match the test to the change	5
Observe decisions and consequences in operation	6
Worked example: a connector change that affects refunds	7
Respond without losing the evidence	8
Make the assurance service sustainable	9
A change review you can run now	10
References	11

Treat release evidence as conditional

An agent passed its release tests last month. Since then, a tool changed its response format, a retrieval collection grew, a provider updated a model and an operational team changed an approval rule. The application still has the same name. The evidence supporting its release may no longer describe the system now performing the work.

Continuous assurance should therefore start with the conditions under which a release decision remains meaningful. Record the workflow, permitted actions, important dependencies, evaluation conditions and operating assumptions. Then identify the changes that could invalidate those assumptions. Monitoring becomes useful when it helps an owner decide whether to continue, constrain, correct or withdraw the workflow.

NIST AI RMF 1.0 treats risk management across the AI lifecycle rather than as a single release event. [1] The practical contribution here is a change-to-evidence map: for each material change, identify the affected behavior, the evidence needed and the authority that can accept the result. The map is a proposed operating tool, not a standard certification scheme.

This approach differs from rerunning every test on a fixed schedule. A full regression suite may be necessary for some releases, but a schedule alone can miss a critical change between runs. Conversely, rerunning a large suite after every cosmetic edit can waste effort

and encourage teams to bypass it. Use the nature and consequence of the change to choose checks, while retaining periodic review for changes that instrumentation may not detect.

The starting question is simple: what would need to remain true for the original release decision to remain defensible? The answer usually includes more than model accuracy. It may include tool permissions, source quality, human review, recovery behavior, workload shape and the availability of a manual fallback. Those conditions become the subjects of ongoing assurance.

Establish the baseline you need to compare

A useful baseline contains both configuration and behavior. Configuration identifies the deployed model or endpoint, prompts, tool definitions, policies, retrieval settings and relevant dependencies. Behavior records what the workflow was expected to do and which evidence supported that expectation. Neither is sufficient alone: a version list does not show acceptable behavior, and a test score without configuration is difficult to interpret later.

Record the business boundary in the baseline. A support agent approved for internal advisers should not automatically be considered ready for direct customer interaction. A drafting assistant should not inherit permission to send messages because the model and prompt are unchanged. Changes to users, authority and context may be more consequential than a model upgrade.

Choose reproducibility information that the team can actually retain. Some hosted systems do not expose a fixed model snapshot or full execution environment. In that case, record the provider-visible version, access date, configuration and observed behavior, and state which aspects cannot be recreated. Do not describe a run as reproducible merely because its prompt was saved.

The evaluation baseline should include representative successes, known difficult cases, important prohibited actions and operational failures. Keep case purpose and expected behavior alongside the input. A case copied from a production incident may become useful regression evidence, but the team should first sanitize it and decide whether the retained material is appropriate for testing.

Anthropic's agent-evaluation guidance distinguishes tasks, trials, grading and execution traces, and discusses assessing both outputs and resulting state. [2] That distinction is useful when designing a baseline: a convincing final message does not prove that a tool completed the intended change, while a correct state change may still have involved an unauthorized intermediate action.

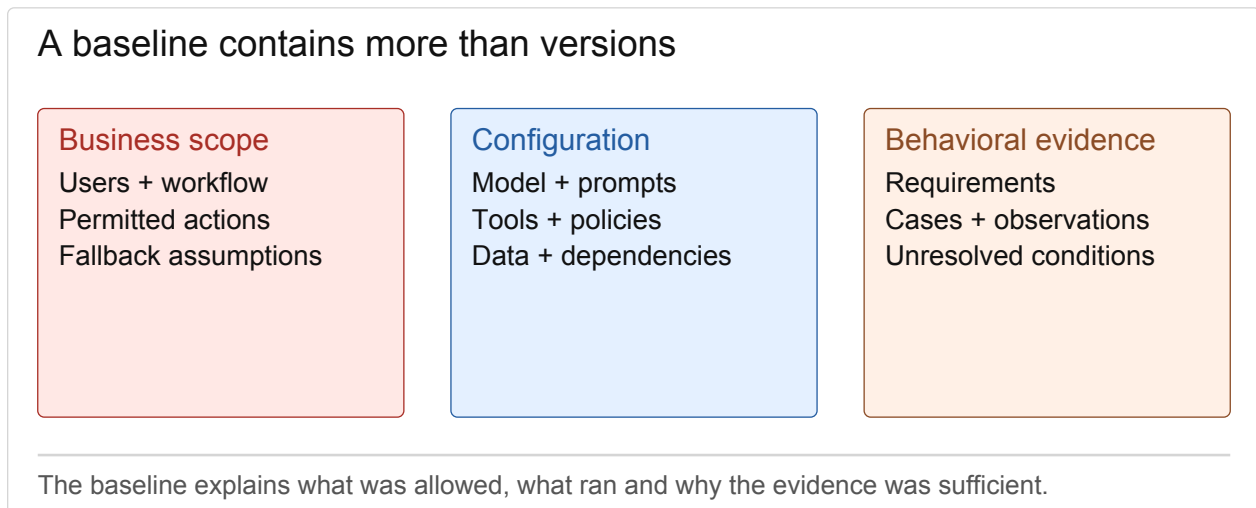


Figure 1. Three baseline views make later changes interpretable. Original Softlander conceptual diagram, 2026-09-16.

Classify changes by what they can invalidate

Use four change classes as a starting point: representation, behavior, authority and operating context. Representation changes alter how information is displayed without intending to change a decision. Behavioral changes affect planning, retrieval, tool use or response generation. Authority changes alter what the agent can access or execute. Context changes alter the users, data, workload or business process around it.

These classes are not mutually exclusive. A new connector may change behavior and authority. A new language may change the operating context and the adequacy of evaluation cases. Classify by potential consequence, not by the apparent size of the code diff. A one-line scope change can be more important than a large visual refactor.

For each change, ask which release assumptions it touches. A retrieval ranking update may affect source selection and answer support. A timeout change may alter retry behavior and duplicate-action risk. A new memory policy may affect later sessions, not just the immediate task. A revised approval rule may change which actions require a human decision.

Then select the evidence that could reveal the relevant failure. A permissions change needs authorization tests. A retrieval update needs checks of access, relevance and source grounding. A tool response change needs parser and workflow tests, including failure paths. A new customer population needs examples that reflect its language, needs and relevant consequences, rather than only more runs of the old dataset.

NIST's Secure Software Development Framework connects secure development practices with the software lifecycle and vulnerability response. [3] Use normal engineering change controls as the foundation. The additional agent-specific work is to trace behavioral and

authority assumptions across components whose outputs can vary and whose effects extend across multiple steps.

A change record should also identify the rollback or containment option. Reverting a prompt may be straightforward; reversing external actions may not be. If the new version has already sent messages, modified records or written persistent memory, recovery needs a reconciliation plan. Do not label a release “reversible” solely because the application can deploy an earlier binary.

Match the test to the change

A compact change-to-evidence matrix helps a team avoid both under-testing and ritual over-testing. For each changed component, record the affected requirement, selected cases, expected evidence, decision owner and residual uncertainty. Keep the matrix small enough to use during a real release review. It should explain why the checks are relevant, not merely list test names.

Use direct deterministic checks where the property permits them. An unauthorized operation should be rejected; a transaction should not be duplicated; a required field should remain valid. Use human or model-assisted assessment for properties that need judgment, such as whether a response addresses the user's question. Keep those methods separate so a subjective quality score cannot conceal a failed permission check.

Repeated trials can help expose variation, but choose the number and interpretation for the question being asked. A small set of successful runs does not establish a rare-event failure rate. If a decision depends on a quantitative reliability claim, design the sample, denominator, uncertainty and acceptable error in advance. This paper provides no universal pass percentage or run count.

Maintain a holdout set where practical, and track which cases have become development targets. If engineers repeatedly optimize against the same visible examples, improved scores may reflect familiarity with that set. Refresh cases based on new failure mechanisms and operating contexts, while preserving historically important regressions. The goal is not novelty for its own sake; it is continued relevance to the release decision.

Choose evidence for the assumption that changed		
Changed element	Selected checks	Decision affected
Tool response	Contract + retries	Transaction integrity
Authority	Permission tests	Allowed actions
Retrieval data	Access + relevance	Supported answers
User population	Workflow review	Appropriate scope

One change may affect several assumptions; combine checks when consequences overlap.

Figure 2. Change classes guide proportionate evidence rather than a single universal test gate. Original Softlander conceptual diagram, 2026-09-16.

Compare against the previous baseline on dimensions that matter. A new version may improve task completion while increasing unnecessary tool calls or making recovery harder. Record tradeoffs rather than collapsing every result into one average. A release owner may accept one tradeoff under a restricted scope and reject it for a higher-consequence workflow.

Observe decisions and consequences in operation

Production observation should connect symptoms to decisions. Latency, cost and error counts matter, but they may not reveal whether a case was correctly resolved or an external action remained within authority. Identify signals for task outcomes, control failures, human intervention and recovery, then determine which can be measured responsibly without excessive collection of sensitive content.

Distinguish a model's assertion from an authoritative outcome. "I updated the record" is not a transaction receipt. "The issue is resolved" is not necessarily a closed business requirement. Where possible, compare the agent's report with the downstream state and record mismatches. Where the outcome cannot be observed automatically, define a sampled review or follow-up process and acknowledge its coverage.

NIST's Generative AI Profile includes ongoing monitoring and risk-management considerations for generative systems. [4] The operating design here turns that principle into a question for each signal: what action should follow if this signal changes? A dashboard with no owner or response condition is visibility without control.

Choose alert thresholds with context. A rise in human escalation can indicate a defect, a new workload or a successful guardrail encountering more difficult cases. Investigate the

composition before treating it as a performance failure. A fall in escalation can also be suspicious if a control stopped triggering. The direction of a metric is not its interpretation.

Make the response proportional. Some signals should trigger investigation, some should constrain a feature and some should stop an action immediately. Separate these outcomes in the runbook. For example, an unusual increase in retrieval misses might route more work to advisers, while a confirmed unauthorized write may justify disabling the affected tool pending investigation.

Avoid promising continuous human review when the operating model does not provide it. Specify coverage, escalation routes and the actions available outside staffed hours. If a workflow needs an immediate response to protect users, build an automatic restriction or choose a deployment scope that the available operating team can support.

Worked example: a connector change that affects refunds

Illustrative scenario. A customer-support agent can recommend refunds and submit approved transactions through a payment connector. The connector changes its timeout and response behavior. The business rule and model remain unchanged. The scenario demonstrates a change review; it is not an account of a real customer system or measured failure rate.

The original release baseline states that each approved refund should create at most one transaction. It also states that an uncertain response must be reconciled before retrying. The connector update is therefore classified as a behavioral change affecting transaction integrity and recovery, even though the user-facing workflow appears unchanged.

The team creates controlled cases for a successful response, a timeout before submission, a timeout after the downstream system commits and a delayed receipt. The test environment records the actual transaction state separately from the agent's final message. A mocked connector is useful for initial cases, but the team also checks the relevant behavior in an authorized integration environment before relying on it.

One constructed test has the connector commit a refund and then time out. The desired behavior is to look up the transaction using a stable reference. If the result remains uncertain, the workflow should hold for reconciliation rather than submit a second refund. Whether the implementation achieves this is a test result to observe; the scenario does not assume success.

The release review examines more than the final answer. It checks the number of downstream submissions, whether approval remained bound to the transaction, which identity executed it and whether the operator can distinguish a completed refund from an unresolved one. A natural-language message saying "refund pending" would not be sufficient if the system had already submitted twice.

The team may release the connector update only for advice-only workflows while transaction checks are unresolved. Alternatively, it can retain the earlier connector version or route refunds to manual execution. These are practical containment choices, not evidence that the new version is generally unsafe. The choice depends on what is known about the specific boundary.

After release, the operating team monitors uncertain outcomes and reconciliation age, alongside transaction records. A newly observed failure becomes a sanitized regression case. The baseline and runbook are updated with the actual finding. This closes the loop between change, evidence and operation without claiming that a test suite can anticipate every future failure.

Respond without losing the evidence

An incident can require immediate containment before the full cause is known. Agree who can disable an action, revoke credentials, reduce autonomy or return work to a manual queue. Preserve the information needed to understand what happened, but avoid copying complete sensitive traces into uncontrolled locations during the urgency of response.

NIST SP 800-61 Revision 3 places incident response within broader cybersecurity risk management. [5] For an agent workflow, connect that discipline to business reconciliation: which actions definitely occurred, which were denied, which remain uncertain and which people or records may need correction? Technical restoration alone may leave unresolved external consequences.

Use a staged recovery decision. First establish containment and a usable fallback. Then identify the affected scope and preserve relevant versions and records. Correct the immediate defect, run the checks tied to the failure mechanism and review whether the original operating conditions still apply. Restore authority gradually when the evidence supports it.

Do not assume that rolling back the model removes harmful persistent state. A memory entry written during the incident may continue influencing later tasks. A queued job may still execute under an old grant. A downstream record may already have changed. Include these paths in the recovery inventory and reconcile them before declaring the workflow restored.

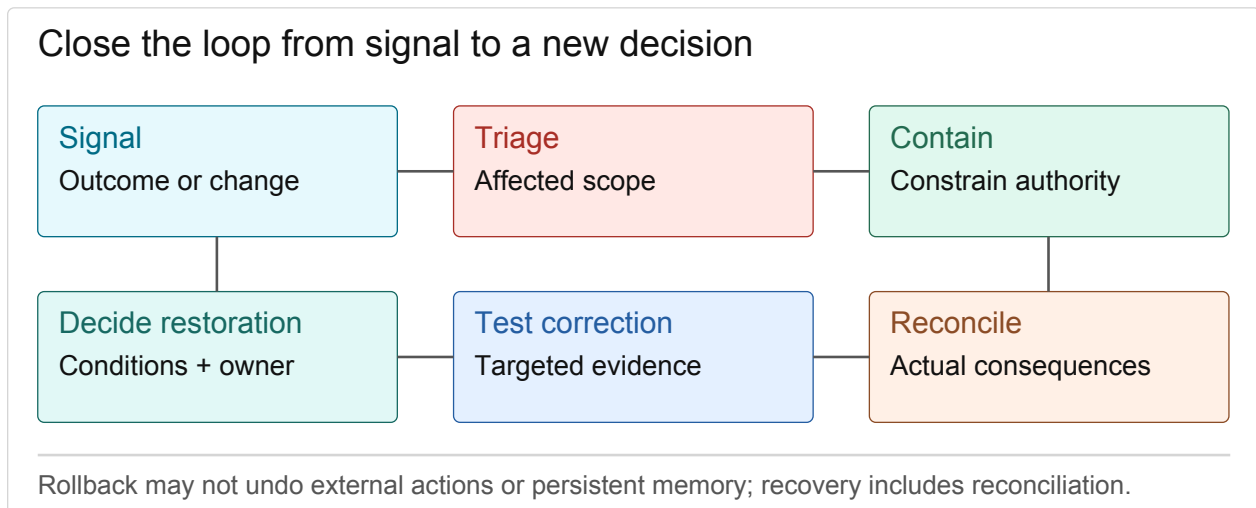


Figure 3. Continued operation depends on an explicit restoration decision after corrective evidence. Original Softlander conceptual diagram, 2026-09-16.

Post-incident learning should produce a specific change. It might be a new test, a narrower permission, a better outcome signal, a revised approval view or a corrected assumption about a dependency. “Improve monitoring” is too broad unless it identifies the event to detect, the responsible owner and the response it enables.

Keep evidence of failures as well as successful corrections. If a remediation introduced another defect or a test did not reproduce the incident, preserve that result. It helps the next reviewer understand the remaining uncertainty and prevents a narrative of inevitable improvement from replacing the actual engineering history.

Make the assurance service sustainable

Ongoing assurance requires a defined operating scope. Identify the workflows covered, the changes reviewed, the signals examined, the cadence, the available response actions and the client responsibilities. A recurring report is not the same as operating a production service, and a periodic review is not the same as continuous incident response. Buyers should be able to distinguish these arrangements before engaging a provider.

Separate engineering ownership from review responsibility. The team changing the system must be able to explain it and provide evidence. A reviewer needs sufficient access and competence to challenge the conclusion. Where independence is required, examine the actual relationship and conflicts rather than using the label as a general description of external advice.

NIST SP 800-53 provides control concepts for configuration management, assessment, monitoring and incident handling. [6] Those concepts can help define responsibilities, but a catalog mapping is not proof that the operating arrangement works. Ask for a demonstration

of the change path, escalation route and evidence handover in the environment that will be used.

Plan for workload variation. A period with many connector, model and policy changes will require different review effort from a stable period. Define what is included in a recurring arrangement and what triggers a separate scope decision. Otherwise, a fixed reporting cadence can conceal a growing queue of unevaluated material changes.

Measure the usefulness of the operating model through decisions and follow-through. Can the team identify which workflows rely on a changed dependency? Can it constrain an affected action promptly? Are uncertain transactions reconciled? Do repeated failures produce revised controls? These questions are more informative than counting the number of dashboards, meetings or test executions alone.

A change review you can run now

Choose the next planned change to an agent workflow and write its intended effect in one sentence. List the baseline assumptions it could disturb. Include authority, data, users and recovery behavior, even if the change request describes only a model or prompt. Identify the accountable person who can accept the resulting operating conditions.

Select evidence before looking at the result. State which cases should remain valid, which new cases are needed and what would require the change to be held or constrained. Keep security invariants distinct from quality judgments. If the decision depends on a numerical threshold, record why that threshold is appropriate for the consequence and what uncertainty remains.

Prepare the operating response at the same time. Name the signal that would reveal the relevant regression, the person or system that receives it and the action available. Check that the fallback is usable and that rollback accounts for persistent state and external actions. Do not wait for an incident to discover that nobody has permission to disable the connector.

After the change, compare the observed behavior with the expected boundary. Record both favorable and unfavorable results, update the baseline and retain the decision rationale. If the evidence is inconclusive, choose a restricted rollout, additional investigation or a hold rather than presenting uncertainty as a successful test.

An agent remains useful only while its behavior, authority and operating context remain acceptable. Continuous assurance is the discipline of noticing when that statement needs to be reconsidered and having the evidence and authority to act. Start with one consequential change and demonstrate that the loop works from request through recovery. Then expand coverage according to the decisions that matter most.

References

- [1] NIST. Artificial Intelligence Risk Management Framework 1.0. January 2023.
- [2] Anthropic. Demystifying evals for AI agents. 9 January 2026.
- [3] NIST. SP 800-218: Secure Software Development Framework 1.1. February 2022.
- [4] NIST. AI 600-1: Generative Artificial Intelligence Profile. July 2024.
- [5] NIST. SP 800-61 Revision 3: Incident Response Recommendations. April 2025.
- [6] NIST. SP 800-53 Revision 5: Security and Privacy Controls. September 2020, with release updates listed by NIST.