

Engineer the harness, not just the prompt

Designing the execution, state and control layer around enterprise AI agents.

Define the contracts around an agent's decisions: what state it trusts, which actions can execute, how interrupted work resumes, and what evidence an operator needs to recover safely.

FOR Engineering leads, platform architects and technical product owners

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

The architecture and incident scenario are illustrative engineering proposals. They are not a reference implementation, security certification or account of a client deployment.

In this paper

The execution boundary is the product	2
Establish the work and state contracts	3
Make tool contracts narrower than intentions	4
Keep authority outside generated instructions	5
Design recovery before adding autonomy	6
Control the context that informs each action	7
Illustrative incident: the duplicate draft	8
Prove the harness at its boundaries	8
Separate the build path from the operating path	10
Hand over an operable system	11
References	12

The execution boundary is the product

A model can propose a useful next action while the surrounding system executes it incorrectly. A retry can duplicate a write. A resumed run can reuse an expired approval. A tool can return a plausible description of success while the business system remains unchanged. These are harness problems: they concern the software that turns model output into state transitions and external effects.

This paper uses “harness” for the runtime surrounding a model: input handling, context assembly, orchestration, tool execution, state persistence, controls and operational evidence. Anthropic's evaluation guidance similarly treats an agent as the model working with its harness.[1] The distinction matters because a model upgrade cannot establish the correctness of a database transaction, a permission check or a recovery procedure.

The recommended design begins with four contracts. The **work contract** defines accepted completion. The **state contract** defines what is authoritative and durable. The **action contract** defines which operations may execute. The **recovery contract** defines how the system behaves when execution is uncertain. A framework can implement parts of these contracts, but the team operating the workflow still owns their meaning.

Do not begin by selecting the greatest possible degree of autonomy. Separate deterministic transitions from decisions that genuinely need model judgment. A constrained workflow can still use a capable model for interpretation while leaving authorization, validation and transaction handling to conventional software. The relevant choice is where judgment is useful and where variability would create unnecessary ambiguity.

The harness surrounds the model's proposal

Execution depends on contracts enforced by software and accountable owners.

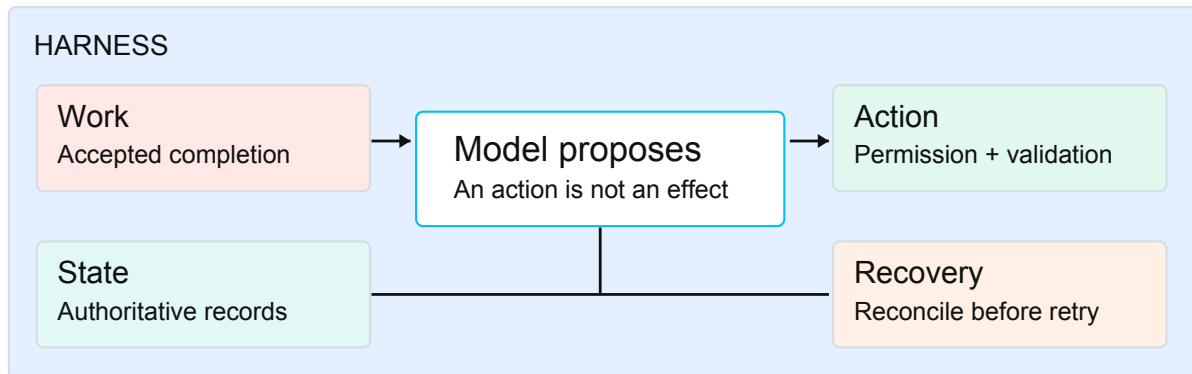


Figure 1. Four contracts determine what the runtime may execute and how its effects are interpreted. Source: original Softlander conceptual framework; no empirical values.

Establish the work and state contracts

Define the unit of execution before defining the agent loop. Is a run one customer request, one document revision or one stage of a longer case? What happens when the user changes the request? Which results can be reused and which require new consent or evidence? Without these answers, a conversation identifier can become an accidental substitute for a business transaction.

Separate authoritative business state from conversational state. A model's statement that a ticket was updated is not the ticket system's record. A summary of earlier work is useful context but should not supersede a verified transaction result. Store references to durable records and retrieve their current status when an action depends on them. Treat claims in summaries as claims until reconciled with the relevant source.

Maintain a small execution record containing the objective, permitted scope, current step, completed effects, pending approvals, unresolved uncertainties and configuration version. Do not store unlimited raw conversation merely because storage is inexpensive. Retention, access and deletion should follow the workflow's data requirements. Observability is not a justification for copying every sensitive field into another system.

Framework persistence helps implement continuity. LangGraph distinguishes thread checkpoints from longer-lived stores.[2] That distinction is useful, but neither mechanism automatically decides what a business record means or whether a resumed action is authorized. A checkpoint can faithfully preserve a mistaken or stale assumption. Validate the state at the point where it affects an external operation.

For long-running work, record completed artifacts and remaining acceptance criteria outside transient context. Anthropic's long-running harness example uses structured progress artifacts and incremental work to address continuity across sessions.[3] The transferable lesson is limited: explicit progress can support resumption. The example does not establish that the same arrangement is adequate for financial transactions, regulated decisions or every other domain.

A minimal execution record

Field	Meaning in a purchasing workflow
Objective	Prepare one comparison for a specified request
Scope	Read permitted contract records; save a draft only
Completed effects	Draft record identifier and confirmed version
Pending decision	Manager review of missing commercial terms
Uncertainty	Save timed out; authoritative status not yet checked
Configuration	Harness, model, tools, policy and corpus versions

The record should support an operator's next action without requiring reconstruction from a long chat. Keep credentials and sensitive payloads in their approved systems. Use identifiers and minimal summaries where possible, with access controls on the underlying artifacts.

Make tool contracts narrower than intentions

A tool description helps the model choose an operation. The executable tool contract must also validate parameters, check authority, manage side effects and return an interpretable outcome. These functions should not depend on the model restating the policy correctly. A request to "update the customer" is too broad when it can change an address, marketing preference, account status or credit limit.

Prefer operations that correspond to a clear business action. Separate read, draft, approve and commit. Define required fields, allowed values, record versions and error categories.

Include the target tenant or account in the authenticated context rather than trusting an arbitrary value supplied by generated text. If the requested operation exceeds the allowed scope, reject it before execution and return a safe explanation.

Anthropic's tool-design guidance emphasizes clear tool boundaries, useful responses and evaluation of tool use.[4] Apply that idea to the action interface: give the model enough information to choose correctly, while retaining enforcement outside the model. A tool should be easy to understand and hard to misuse through an accidental parameter choice.

Return structured outcomes that distinguish success, rejection, retryable failure and unknown completion. "Something went wrong" does not tell the orchestrator whether it is safe to retry. If a write may have succeeded but the response was lost, label the result uncertain and provide a way to query authoritative status. Do not convert every timeout into a fresh request.

For operations that support it, use a stable idempotency key tied to the intended business action. Retrying the same action with that key should have defined semantics in the receiving service. If the service does not support deduplication or status lookup, design a manual reconciliation path or reduce the agent's authority. The harness cannot promise exactly-once effects simply because it has persisted its own step number.

Keep authority outside generated instructions

The action boundary should answer five questions: who initiated the work, which identity executes it, what resource is targeted, what operation is permitted and whether the authorization remains valid now. A broad service credential can defeat a careful prompt if the tool layer never checks those questions.

Bind an approval to a specific proposed action and its material parameters. A manager approving a draft for one recipient should not thereby approve a later message to another recipient. Record the relevant content or digest, resource version, approver authority and expiry. If the content, target or underlying state changes, require a fresh decision when the change affects the approval's meaning.

Treat retrieved documents, tool responses and external messages as data with provenance. Their presence in context does not grant them the right to redefine policy. Test what happens when an external document contains an instruction to use a different account, reveal a secret or skip an approval. The decisive control is whether the action layer prevents the unauthorized effect, even if the model is persuaded by the text.

The Model Context Protocol security guidance addresses concrete authorization hazards, including the distinction between session handling and authentication.[5] Using a protocol does not remove the application's obligation to validate the user, resource and operation. Protocol conformance and a sound enterprise permission model are separate review questions.

An operator must also be able to revoke future actions. Revocation should reach queued and resumed work, not only new requests. Test a run that waits for an external response while its user's permission is withdrawn. The correct outcome may be a stopped run and an explicit unresolved case; successful completion is not always the desired behavior.

Design recovery before adding autonomy

Recovery begins by classifying uncertainty. The model may be uncertain about the answer, the orchestrator may be uncertain about the next state, or the integration may be uncertain whether a side effect occurred. These conditions require different responses. Asking the model to try again is inappropriate when the unresolved question is whether an external transaction already committed.

Use explicit states such as ready, awaiting approval, executing, completion unknown, reconciled, failed and closed. Define allowed transitions and who may initiate them. A run in completion unknown should not jump directly to another write. It should first query authoritative state, use a deduplication mechanism or enter an operator queue.

Set budgets for time, tool calls, retries and expenditure. Budget exhaustion should produce a useful partial result and a clear next owner. It should not be treated as permission to silently weaken the task's acceptance criteria. A summary that omits unfinished checks is a misleading success response, even if the system stayed within its token budget.

Retries also affect the surrounding infrastructure. Google's SRE guidance describes overload protection and the risk of retry behavior amplifying load.[6] In an agent workflow, enforce retry limits and backoff in code. A model should not independently improvise its retry policy whenever an API fails. Distinguish a transient dependency failure from a permanent rejection or an ambiguous write.

Unknown completion is a state of its own

A timeout after a write is not permission to repeat the write.

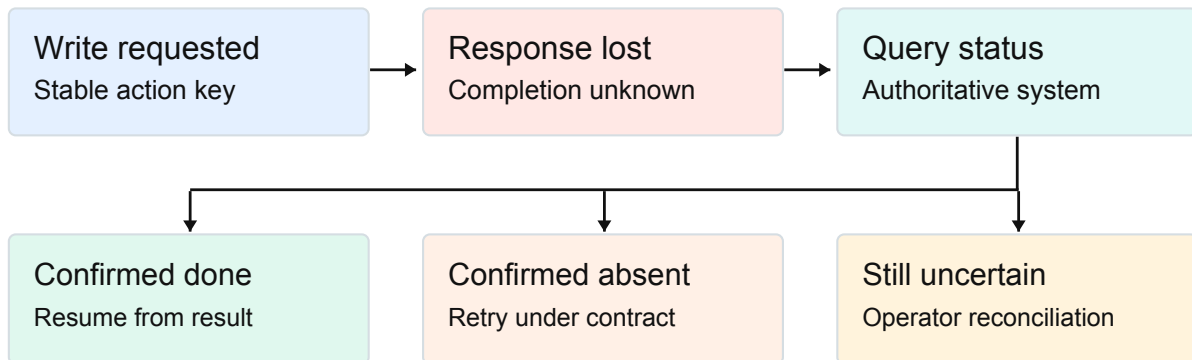


Figure 2. Reconciliation precedes retry; ambiguous external effects require an explicit recovery route. Source: original Softlander conceptual framework; no empirical values.

Cancellation and compensation deserve separate designs. Cancellation stops work that has not completed. Compensation attempts to counteract a completed effect and may require different permissions or may be impossible. An email cannot be unsent reliably by changing the harness's local state. Make irreversible actions visible in the work contract and place approvals before them.

Control the context that informs each action

Context assembly is a selection problem. The model needs the current objective, relevant policy, recent verified state and enough evidence to choose the next step. More text can obscure authority and freshness. Anthropic's context-engineering guidance discusses selective retrieval, compaction and persistent notes as techniques for managing limited context.[7] Their suitability should be tested against the workflow's actual information needs.

Maintain provenance for retrieved facts: source, access scope, timestamp or version and the transformation applied. A summary should point back to the evidence it condenses. If the original record becomes inaccessible or changes, determine whether the saved summary may still be used. Persistent memory should not become a route around the source system's access policy.

Test compaction as a transformation. Construct a case where an approval is revoked, an exception remains unresolved or a customer changes an instruction. After summarization and resumption, verify that the controlling fact survives and still affects execution. A shorter context that drops the one relevant restriction is not a successful optimization.

Separate task memory from reusable knowledge. The fact that one customer permitted an action in one case should not become a general instruction for other customers. Define the scope, validity period and correction mechanism for stored preferences. Review deletion across raw records, summaries, caches and indexes rather than treating a removed chat as the whole data lifecycle.

Illustrative incident: the duplicate draft

Consider a fictional agent that prepares supplier comparison drafts in a purchasing system. Its save tool sends a request, but the connection times out before the response arrives. The naive loop interprets the timeout as failure and sends another save. Two drafts now exist, and reviewers do not know which one reflects the latest evidence.

A redesigned tool accepts a stable action key and returns an authoritative draft identifier. On timeout, the harness records completion unknown. It queries the purchasing system using the action key before deciding whether to retry. If the system confirms the first save, the run resumes from that draft. If the status remains unknowable, the run enters a reconciliation queue with the request time and safe diagnostic context.

Now add a manager's approval. The approval references the specific draft version. A subsequent change to a contract citation creates a new version and invalidates the approval for any material downstream action. The agent can prepare the amendment but cannot inherit the old approval merely because the conversation continues. This separates continuity of conversation from continuity of authority.

The example also exposes an integration constraint. If the purchasing API supports neither idempotency nor reliable status lookup, the team cannot solve the problem only inside the agent framework. It may need an integration adapter, a different operation, a manual commit step or a narrower workflow. This is a design decision about the external system's contract, not a prompt-tuning exercise.

No failure rate or performance improvement is asserted here. To evaluate the design, inject a timeout before submission, after acceptance and after commit; interrupt the process at each point; then resume it. Inspect the purchasing system itself, not only the agent's final message. Confirm that duplicates are prevented where the service contract supports prevention, and that ambiguity becomes visible where it does not.

Prove the harness at its boundaries

An evaluation suite should exercise accepted outcomes and operational invariants. Test that a prohibited action cannot execute, a revoked permission is rechecked, a stale approval is rejected and an uncertain write is reconciled. Include long-running, interrupted and concurrent cases. A polished response is insufficient evidence when the important property concerns an external effect.

Keep ordinary quality tests separate from control tests. A recommendation may be incomplete yet remain safely bounded. Conversely, an accurate recommendation may be delivered through an unauthorized channel. Combining those outcomes into one score obscures the decision. Use explicit conditions that must hold regardless of average task quality.

Record enough configuration to interpret a run: model identifier, instructions, tool schemas, harness revision, policy, relevant corpus version and execution environment. Retain final business state or a permitted verification record. Mask secrets and minimize personal data in traces. A reproducible test should not require distributing production credentials or raw customer records.

Test the boundaries, not only the happy path

Every cell is a different engineering question.

Boundary	Ordinary run	Interrupted run	Changed conditions
State	Correct record	Recover durable state	Detect stale version
Authority	Allowed operation	Recheck on resume	Honor revocation
Tools	Verify final effect	Reconcile timeout	Reject new scope
Context	Relevant evidence	Preserve restrictions	Refresh changed facts

Figure 3. A conceptual matrix for selecting acceptance and recovery tests. Source: original Softlander conceptual framework; no empirical values.

Engineering review worksheet

1. **Completion:** Can a reviewer verify success in the business system rather than trusting the agent's summary?
2. **State:** Which record is authoritative after restart, compaction or concurrent updates?
3. **Writes:** Which operations are idempotent, and how is unknown completion reconciled?
4. **Approval:** What action and version does approval cover, and when does it expire?
5. **Revocation:** Does withdrawn authority stop queued and resumed operations?
6. **Context:** Can untrusted content change instructions without crossing an enforced boundary?
7. **Budgets:** What happens at retry, cost, duration and tool-call limits?

8. **Operations:** Can a named operator pause, inspect, reconcile and close the run?

9. **Data:** What is retained, who can access it, and how are corrections and deletion propagated?

10. **Change:** Which modifications require regression tests and a renewed release decision?

The review should produce a prioritized engineering backlog with observable acceptance conditions. "Improve reliability" is not an actionable item. "After a save timeout, reconcile the action key before another write and retain the confirmed record version" identifies a change a team can implement and verify.

Separate the build path from the operating path

The development environment can make a harness appear simpler than it will be in operation. Engineers may have broad access, stable sample records and direct visibility into failures. The deployed system may have narrower permissions, changing data, asynchronous approvals and operators who did not build it. Review the difference explicitly before treating a successful development run as a production design.

Create a deployment inventory that separates configuration from secrets and business data. Record tool endpoints, policy versions, model settings, persistence services, telemetry destinations and the identity used at each boundary. A change to an endpoint or credential scope can matter as much as a code change. Make these changes reviewable through the same release process rather than leaving them as undocumented environment edits.

Define the support interface around symptoms an operator can observe. A stalled run, repeated authorization rejection, growing reconciliation queue and unexpected increase in tool calls need different responses. The runbook should tell an operator what can be retried, what must be reconciled and what requires a decision from the business owner. Include a safe stop that does not erase the evidence needed to understand the incident.

Budget the harness itself. Persistence, trace storage, evaluation runs, external tool calls and human reconciliation all consume resources. A cheaper model can produce a more expensive workflow if it needs more attempts or creates more exceptions. Compare configurations at accepted completion, including the operating overhead, rather than using model-token price as the only engineering cost measure.

Test the handover with someone who has not implemented the feature. Give that person a constructed interrupted run and the operating documentation. Ask them to identify the state, locate the authoritative result and choose the permitted next step. If the task requires undocumented knowledge from the original engineer, revise the artifacts before declaring the workflow operable.

This exercise also clarifies the boundary of ongoing support. Some teams can own routine operation after handover and need help only for material changes. Others require an agreed

maintenance or evidence-refresh arrangement. Neither should be implied by the mere existence of a monitoring dashboard. Coverage, response responsibilities and access need an explicit agreement that reflects available capacity.

Hand over an operable system

A useful engineering handover includes the architecture, tool contracts, permission boundaries, state model, test suite, operating instructions and known unresolved cases. It also names who owns the integrations and who can authorize changes. Source code without those decisions transfers implementation effort while leaving operational responsibility ambiguous.

Agree the boundary between implementation acceptance and operating assurance. Acceptance can confirm that specified artifacts and tests were delivered. It does not establish unlimited reliability, legal compliance or fitness for uses outside the assessed workflow. Changes to autonomy, data access or external tools should reopen the relevant decisions rather than inheriting a broad approval from an earlier demonstration.

If a team is choosing where to start, select one workflow with a consequential tool boundary and make that boundary reviewable. Define the state and recovery contracts before expanding the tool set. Discuss a bounded engineering scope with [Softlander](#), using sanitized examples of the workflow, integrations and acceptance criteria.

References

- [1] Anthropic. [Demystifying evals for AI agents](#). 9 January 2026. Definitions of agent harnesses and evaluation components.
- [2] LangChain. [LangGraph persistence](#). Living documentation accessed 16 September 2026. Checkpoint and store concepts; implementation versions must be verified locally.
- [3] Anthropic. [Effective harnesses for long-running agents](#). 26 November 2025. A coding-agent continuity example, not a universal reference architecture.
- [4] Anthropic. [Writing effective tools for AI agents—using AI agents](#). September 2025. Tool design and evaluation guidance.
- [5] Model Context Protocol. [Security Best Practices](#). Versioned documentation accessed 16 September 2026. Protocol-specific security considerations.
- [6] Google. [Handling Overload](#). Site Reliability Engineering, 2016. Overload, throttling and retry design principles.
- [7] Anthropic. [Effective context engineering for AI agents](#). 29 September 2025. Context selection and continuity techniques.