OMNAFY

# Earned autonomy

Risk tiers, approval queues, and how supervision ends

| | |
|---|---|
| Audience | Operations leaders |
| Version | 1.0 |
| Published | August 31, 2026 |
| Online | omnafy.com/whitepapers/earned-autonomy/ |

## Abstract

Every organization that puts a human in the loop discovers the same two failure modes, usually in this order. First the queue is a bottleneck, because every action waits on a person and the automation saves nobody any time. Then the queue is a rubber stamp, because the reviewer has seen four hundred correct proposals in a row and is now clicking release in under three seconds. The second failure is worse than the first, because it looks like success.

This paper describes how Omnafy is designed against both. Risk is classified per tool rather than per caller, so the volume of work that reaches a queue is proportional to actual consequence and read-only operations never enter one. Approvals carry mandatory evidence, so a decision is made from the record rather than from trust. And autonomy is earned rather than configured: a specific agent identity, on a specific task, graduates out of the queue on measured evidence with a human signature, and that graduation is reversible without anybody's intervention when quality slips.

One category never graduates. Tools whose substance is professional or legal judgment are tier T3, and their permanence is enforced in three independent places in the system rather than written in a policy document. At our home health design partner, the clinical decision to admit or decline a referral is T3. The agent assembles the chart and recommends with evidence. A clinician decides. That never changes, not after a thousand clean runs.

## 1. The supervision problem

Supervised automation is sold constantly and delivered rarely. The reason is that supervision has a cost curve nobody prices in at the demo.

At the start, every proposed action is reviewed and the reviewer is careful, because everything is new and some of it is wrong. This phase produces genuine value: mistakes are caught, and the corrections are information about where the automation is weak. It also produces very little throughput, because a person is in the path of every action.

Then the automation gets good. Proposals become correct almost all the time. The reviewer's attention, which is a finite resource, is now being spent on a stream of items that are nearly all fine. Attention degrades to pattern-matching, pattern-matching degrades to clicking, and the queue quietly stops being a control. The organization still believes it has human oversight, because the screen still exists and someone is still clicking.

The worst part is what happens next. If approval history is being used as evidence that the automation works, then a rubber stamp is being laundered into a quality signal. The record now measures the reviewer's attention span rather than the agent's accuracy, and any decision made on that record is made on a corrupted input.

Three design choices follow from taking that curve seriously.

**Volume must be proportional to risk, not to activity.** If reads go through a queue, the queue fills with items that carry no consequence and trains everyone to skim.

**A decision must be possible from the record alone.** If deciding correctly requires opening another system to check something, most reviewers will not check, most of the time.

**Supervision has to end somewhere, on evidence, reversibly.** A system that never removes an approval step is a system that guarantees the rubber stamp eventually. The design goal is that demonstrated-safe work stops consuming attention so that attention concentrates where it still matters.

## 2. Risk tiers

Risk is a property of the tool, not of the caller. A tool's tier answers what the worst plausible consequence of this operation is, which is the same regardless of who calls it. The caller contributes a ceiling, meaning the highest tier it may reach at all.

The tier is declared in the tool's manifest and ratified by a human when the tool server registers. We never take a tool server's word for its own risk, and ratification can raise a declared tier but never lower it.

| TIER | NAME | COVERS | DEFAULT POSTURE |
|---|---|---|---|
| T0 | Observe | Read-only monitoring, reporting, retrieval, listing | Autonomous immediately |
| T1 | Internal actions | Reversible writes inside your own systems | Propose then approve, until graduation is earned per identity and task |
| T2 | External actions | Anything leaving your boundary: external records, faxes, messages | Approval-first for longer, graduating selectively |
| T3 | Judgment calls | Professional or legal judgment | Permanently human |

### 2.1 T0, and why reads do not queue

Reads are autonomous immediately for any principal whose scope grants the tool. No approval, no graduation, no queue.

That is a deliberate position, and it is the first place a cautious buyer pushes back. Information disclosure is a real risk, so why is there no review step?

Because it is a scoping problem rather than an approval problem. Three other mechanisms already carry it. Scope decides which reads a principal can perform at all, which is the minimum-necessary control. Visibility filtering means a model never learns of reads outside that scope, so it cannot be steered toward one. And every read still emits an audit event, so what was read by whom is queryable.

Routing every read through a human queue would add no control that scope does not already provide, and it would flood the queue with consequence-free items. That is the fastest way to teach approvers to rubber-stamp, which is the thing this entire design is trying to prevent.

### 2.2 T1, where autonomy is earned

Reversible writes inside your own systems: updating a referral's triage status, creating a draft chart entry, tagging a document, drafting a workflow definition.

The tier's premise is that internal reversible writes are exactly the category where supervised autonomy is safe to earn. Mistakes are visible inside your own systems and can be undone, so a history of correct proposals is meaningful evidence rather than survivorship.

Two boundary cases are resolved at ratification rather than at run time. A write that is internal but not reversible, for example a submission that triggers an irreversible downstream workflow, should be ratified T2 despite being an internal write. And a write whose reversibility depends on timing is treated as irreversible.

### 2.3 T2, where the bar is higher

Anything that leaves your boundary: filing to an external records system, sending a fax, messaging a third party.

External actions cannot be rolled back by you. A fax cannot be unsent, and the blast radius includes third parties. So the evidentiary bar is higher: larger evaluation windows, stricter thresholds, and graduation extended only to narrow, high-volume, low-variance tasks. Many T2 tasks reasonably never graduate. Our position is that this is a per-task business decision backed by evidence rather than a platform default in either direction.

### 2.4 T3, which never graduates

Operations whose substance is professional or legal judgment. T3 is not a slower T2. It is a different kind of claim. Not "this action is risky enough to need review for now" but "the decision itself belongs to a licensed or accountable human, categorically."

Ratifying a tool at T3 is your organization stating that no history of agreement, however long, converts this decision into an automation target.

The permanence is structural, not configurational, and it is enforced in three independent places. The policy plane has no representable graduation state for a T3 tool, so the machinery that would promote a pair simply does not admit T3 entries and there is nothing to misconfigure. Non-human principals are ceiling-capped at T2 by construction, so even a corrupted policy state that invented a T3 graduation entry would meet a caller that cannot clear the ceiling check. And visibility filtering never lists a T3 deciding form to a non-human principal, so a model cannot be steered toward a tool it has never seen.

The canonical example, from our home health design partner: upstream of the decision, agents do substantial work at lower tiers. An intake agent ingests the referral documents, triages them, assembles the chart, and interprets eligibility, all T0 reads and T1 proposals. Its output at the decision point is a recommendation with evidence attached: the assembled chart, the extracted referral facts, the eligibility interpretation, identified risk factors, and the recommendation with its reasoning. Producing and submitting that recommendation is itself only a T1 internal write. The deciding tool is ratified T3, its deciding form accepts only human principals, and the clinician's decision is recorded under the clinician's own identity with the full chain showing which agent recommended and which clinician decided.

## 3. Approval queues

The design goal, stated by the operations leaders this is built for: an approver should be able to make a correct decision from the record alone, in seconds, with no side-channel investigation, and the record of that decision should be durable evidence for graduation.

### 3.1 What a queue item is

When the call-time policy check returns pending-approval, two artifacts are created on opposite sides of the deployment boundary.

Inside your account, the held call and its evidence bundle. The full proposed call, meaning the tool and its arguments and the principal chain, is persisted alongside the evidence bundle assembled per the tool manifest's evidence hints: the inputs that led here, retrieved context, the proposed action rendered for

review, expected effects including whether the action is reversible, and expected cost. Payloads never leave this side.

In our control plane, the queue item. It holds the redacted projection: item id, tenant, principal chain, namespaced tool, tier, side-effect class, cost estimate, routing state, expiry deadline, and an opaque reference to the evidence bundle. Never the bundle itself.

### 3.2 Surfacing evidence without moving it

When an approver opens an item in the console, the console does not fetch the evidence from our infrastructure, because we do not have it.

The console resolves the evidence reference directly against your deployment. That fetch is itself a governed read through the gateway, made under the approver's own identity, so it is scope-checked and audited like any other read. An approver can only view evidence their scope reaches, and even the act of reviewing crosses the one door and leaves a record. The payload travels from your boundary to the approver's browser and never transits our infrastructure.

One operational consequence follows: an approver whose scope cannot reach the evidence cannot meaningfully review the item. Routing configuration is validated against approver scopes to prevent exactly that mismatch.

### 3.3 Routing

Items route to named queues by configuration, evaluated in order of specificity. The proposing principal's grant may pin routes per tier. The queue configuration may map patterns of namespace, tier, and task to queues. And a tenant-level fallback queue catches anything unmatched.

The fallback is mandatory. An item that matches no route lands somewhere visible rather than being dropped, because a silently vanishing proposal is a fail-open failure in disguise.

Routing changes are administrative mutations: tiered, approved, and audited like any other operation.

### 3.4 Expiry and escalation

Every item carries a deadline from its queue's configuration.

**Expiry is a denial.** The held call is discarded, an expired decision event is written, and the proposing run is notified. An unattended queue converges to denials, never to timeouts-as-approvals. Fail closed, applied to humans.

Escalation fires before expiry, at configured thresholds: the item is re-routed to an escalation queue or additional approvers are notified, with its evidence reference and full routing history intact. Escalation exists so that queue neglect surfaces as an operational signal while the work can still be saved. Expiry exists so that neglect cannot silently become an approval.

### 3.5 Two verbs

An approver has exactly two verbs: release or reject.

There is deliberately no edit-then-release. A modified action would be the approver's action wearing the agent's attribution, and it would poison the evaluation semantics, because graduation measures whether the agent's proposals are correct as proposed. If a proposal is wrong, rejecting it with a reason is the useful signal, and the rejection reason frequently steers the agent's next step.

A rejection carries a structured reason code and an optional note. Reason codes are what make rejections legible to the evidence machinery later.

On release, the decision flows down with the approver's identity, and the gateway re-checks policy before dispatching the held call. The principal may have been disabled, the tool's manifest may have changed, which voids its ratification, or the kill switch may have fired since the item was queued. **A release is a necessary input to execution, never a bypass of enforcement.**

### 3.6 A T2 approval, end to end

The intake agent proposes a records-request fax to a referring physician's office.

The agent calls the tool. The gateway resolves the manifest, sees T2 and external-action, and the posture evaluates to pending-approval. It persists the held call and evidence bundle inside your account, posts the redacted item to our queue, and returns a structured pending result to the run, which parks rather than holding compute open against a human's response time.

The item routes to the intake supervisor queue. The approver opens it, and the console resolves the evidence reference back through the gateway under the approver's own identity, so the evidence travels from your boundary to their browser directly.

The approver releases with a recorded reason. The decision flows down to the gateway, which re-checks policy and dispatches the held call to the communications adapter. Full-fidelity audit events land in your account, and redacted envelopes cross to us. The scheduler relaunches the parked run, which replays its journal and resumes with the result.

Had the approver rejected, the held call is discarded and the same two audit writes happen with outcome rejected. Had nobody acted by the deadline, expiry produces the same discard with outcome expired. In all three endings the run learns the outcome and may adapt, because a rejected action fails the action rather than necessarily the run.

## 4. Designing against approval fatigue

A queue that trains its approvers to rubber-stamp is worse than no queue, because it launders unreviewed actions into approved evidence. Five mechanisms work against that.

**Evidence schemas make the reviewable material the default presentation.** Reviewing is easier than not reviewing, which is the only version of this that survives contact with a busy day.

**T0 never enters the queue**, so volume stays proportional to actual risk.

**Graduation removes demonstrated-safe pairs from the queue entirely**, so the queue trends toward the items that still need a person rather than growing without bound.

**Queue-health metrics are surfaced in the console**: median decision latency, per-queue expiry rate, per-approver release rate. A queue drifting toward instant release on everything is visible to you before it is a finding in someone else's audit.

**Sampled re-review means a rubber stamp is never the last word.** After a pair graduates, a configured fraction of its autonomous actions is copied back to its former queue as retrospective review items, on a schedule the pair cannot observe.

The residual is real and we state it rather than claiming a fix. This is a human-process threat and no mechanism eliminates it. The design goal is that inattention is visible in the metrics and never final because of sampling and demotion. The failure mode that remains is an organization that ignores its own queue-health dashboard.

## 5. Autonomy graduation

Graduation is the answer to the question every approval-first system eventually faces: when does supervision end?

The answer: per identity and task, on evidence, with human sign-off, reversibly, and never for T3.

### 5.1 The unit is an identity and a task

Graduation never attaches to a whole agent ("the intake agent is trusted now") or to a whole definition ("version 4 is trusted now"). It attaches to a pair.

The identity side, for agent callers, is the durable agent identity shared by every run of a definition. Evidence accumulates across runs, and demotion strikes all future runs at once. For human and service principals the identity side is simply the principal, and the mechanism is type-agnostic, though in practice customers grant human principals standing autonomy at onboarding for tools already within their job authority while agent identities earn it from history.

The task side is a declared name. A definition enumerates the tasks it performs as part of its versioned content, the runtime stamps every proposed call with the task it is acting under, and the id travels in the queue item and the audit event. Task ids are stable across definition versions, which is what lets evidence accumulate, but a new version that materially changes a task's instructions triggers re-certification for every pair involving that task. A proposal bearing no task id, or an undeclared one, is a fail-closed denial.

Extracting referral header fields from an inbound document is one task. Proposing a chart assembly is another. They graduate independently even when one agent definition performs both, because their error profiles have nothing to do with each other.

### 5.2 Evaluation sets

The evidence for graduation is the pair's own supervised history. Every decided queue item is a labeled case: the evidence bundle is the input, and the release or rejection with its reason code is the label.

An evaluation set is a curated selection from that history, sampled across time and input variance, excluding cases decided under superseded task instructions, at a minimum volume set by threshold policy.

Placement follows the data. Evidence bundles contain regulated data, so evaluation sets live and evaluate inside your boundary. Our control plane holds the thresholds and receives only the resulting metrics, meaning case counts, release rates, and reason-code distributions, as redacted values across the standard upward channel. The residency invariant holds for the graduation machinery exactly as it does for everything else.

### 5.3 Thresholds

Thresholds are policy, set per tenant and per queue, and stricter for T2 than for T1. The defaults below are configurable and are stated here to fix the shape rather than to prescribe a number.

| PARAMETER | T1 DEFAULT | T2 DEFAULT |
|---|---|---|
| Minimum decided proposals in window | 50 | 200 |
| Release rate over trailing window | at or above 98% | at or above 99.5% |
| Trailing window | 60 days | 120 days |
| Incident-class rejections in window | 0 | 0 |
| Task-instruction stability required | full window | full window |

A pair that clears its thresholds becomes a graduation candidate. Nothing happens automatically at candidacy. Crossing a statistical bar creates an item for a human, never a state change by itself.

### 5.4 Promotion

Promotion requires sign-off by a human holding the graduation-administration scope, typically the owner of the queue the pair has been routing to, since that person has been the pair's supervisor. The sign-off reviews the metrics, the sampled cases behind them, and the blast radius of the tool involved.

The promotion is a policy-state change and is audited as one: an event carrying the pair, the evaluation metrics as of promotion, the threshold policy version, and the signing principal. The new state replicates to the gateway, and from the next proposal onward the pair's calls at the graduated tier evaluate to allow instead of pending-approval.

Nothing else changes. Scope, ceiling, audit emission, and metering are untouched. **Graduation removes a wait, not a control.**

### 5.5 Demotion is automatic

Promotion needs a human. Demotion deliberately does not. The asymmetry is the point: the platform is biased toward supervision.

Any of the following returns a pair to propose-then-approve immediately, with an event recording which trigger fired.

An error-rate trigger, when tool-call failure or downstream-error rate for the pair exceeds its threshold. A re-review rejection trigger, when sampled re-review rejects more than the tolerated fraction of the pair's autonomous actions. An incident trigger, meaning any incident flag naming the pair, its definition, or its tool, including a kill-switch exercise. A change trigger, when a new definition version materially changes

the task's instructions, or when the manifest of a tool the task calls changes, which already voids that tool's ratification and cascades here. And a certification lapse, when the re-certification deadline passes without a completed check. A lapsed certificate demotes: fail closed, applied to trust itself.

Demotion takes effect at the next proposal. In-flight autonomous calls complete under the state they were admitted under and are prime candidates for re-review. Re-promotion is not a shortcut: the pair re-enters candidacy only by clearing its thresholds again, on evidence accumulated after the demotion.

### 5.6 Re-certification and sampled re-review

Graduation decays unless renewed. On a fixed cadence, quarterly for T1 pairs and more frequent for T2, each graduated pair's evidence check re-runs over the current trailing window. Passing is a non-event: recorded, changing nothing. Failing or missing it demotes.

Between certifications, sampled re-review keeps the evidence honest. A configured fraction of each graduated pair's autonomous actions is copied to its former queue as retrospective review items, with the same evidence presentation but no held call, since the action has already executed. Their outcomes feed the evaluation set and the re-review rejection trigger.

Sampling is what defends the whole construction against evidence gaming. A pair that learned to look good under supervision is still being examined, on a schedule it cannot observe, after supervision ends.

### 5.7 The state machine

A pair begins in propose-then-approve on its first activity. Clearing thresholds over the trailing window makes it a candidate. Human sign-off moves it to autonomous, and a declined sign-off or stale evidence returns it to propose-then-approve. Re-certification passes keep it autonomous. Any demotion trigger, or a failed or lapsed re-certification, returns it to propose-then-approve.

T3 pairs never enter this machine. The policy plane does not represent graduation state for T3 tools, so the initial state does not exist for them.

Every transition, including a re-certification pass that changes nothing, is an audited event. There is no silent path between autonomy states.

## 6. The controls around all of it

Three more mechanisms belong in an operations leader's mental model, because they are what you reach for when something is wrong.

**The kill switch.** Halt one run, or every in-flight run of a definition. It is exercised by an authorized person through the administrative API and lands as an audited mutation. Semantics are immediate cessation rather than graceful shutdown, so a tool call already dispatched may complete in the wrapped system: the kill switch stops the agent, it does not unwind the world. The unresolved call is recorded as such, and the tool's reversibility metadata is what tells you whether cleanup is possible. The run moves to halted with the halting person recorded, which is the one terminal state that names a human. Its transcript is sealed exactly as for any other terminal, which is usually why it was halted.

The kill switch changes run state, never definition state. Stopping future runs is a lifecycle act: retire the version or disable its trigger. The operational response to a misbehaving definition is halt-all plus retire, two audited acts by an accountable person.

**Budgets and timeouts.** Every run is bounded on three axes fixed at admission and not adjustable mid-run by anyone including the agent: a monetary budget covering model spend and tool cost, a tool-call budget covering total calls including sub-agents, and a wall-clock timeout on active execution. Checks are pre-flight, so a step whose estimated cost the remaining budget cannot cover is not started, and an in-flight call is never severed for budget reasons, because a budget mechanism that truncates writes mid-flight converts a cost control into a data-integrity hazard. Time spent parked awaiting approval does not count against the timeout, so a slow queue surfaces as a queue problem rather than silently consuming run budget and manufacturing pressure to approve quickly.

Raising a budget means a new definition version through the full lifecycle. Repeated budget exhaustion for one definition is an operational signal that the definition should be redrafted and re-approved, not overridden at run time.

**Versioned definitions.** Agent behavior is a declarative, versioned artifact, so a behavior change is a diff a human approved. Drafts are the only mutable state; anything approved is frozen and a change is a new version through the full lifecycle. At most one version is live at a time, and activation is atomic: the new version goes live and the previous one retires in one transition. Reverting is a new version carrying the prior content through review again, because a revert that skipped review would be an unreviewed change wearing an old version number.

An agent may draft a definition. Only a human may approve one. No drafting history, no graduation, and no ceiling configuration ever lets the drafter approve their own definition.

## 7. The reference workload

Our design partner is a home health agency, and their referral intake pipeline is the workload the automation module was designed against. Referrals arrive as faxes and emails. Each must be classified, matched or filed against a patient record, assembled into a chart, interpreted for eligibility, reviewed

clinically, and, if admitted, scheduled.

The whole pipeline executes inside their own cloud account. Here it is stage by stage, with the tier of the actions each stage takes.

**Ingestion and classification.** Inbound documents fire a trigger, and a run reads and classifies them, which is T0, then files each referral into the intake queue, which is a reversible internal write at T1. Early in the deployment those T1 writes run propose-then-approve. As approval history accumulates, individual pairs of identity and task graduate to autonomy, reversibly.

**Triage and patient matching.** The run matches the referral against existing patient records and proposes new-versus-existing routing. Same T1 posture: propose, approve, earn autonomy per task.

**Chart assembly.** The run assembles the chart in the clinical system. Internal reversible writes at T1.

**Eligibility interpretation.** The run interprets coverage and eligibility and produces a recommendation with evidence: inputs, retrieved context, reasoning, proposed disposition. It takes no consequential action of its own here. Its output is the evidence package for the next stage.

**Clinical review.** Admit or decline is a professional judgment call. T3, permanently human. A clinician decides from the evidence, the agent never does, and this never graduates. The decision lands in the audit trail with the full chain: which run assembled the evidence, which clinician decided.

**Scheduling.** For admitted patients, proposing visit schedules inside their own systems is T1. Anything that leaves the boundary, meaning confirmation faxes or messages to patients, is T2 and requires communications adapters, which are designed and not yet shipped. Those outbound steps remain human-executed today.

Two properties of this deployment generalize. The tier structure maps a real regulated workflow without forcing it: mechanical work graduates toward autonomy and judgment stays human by construction. And everything the pipeline touches stays inside their account, with our control plane seeing only redacted envelopes.

## 8. What we have not proven yet

Three limits belong in front of an operations leader before a pilot rather than after one.

**No per-tenant spend governor.** Per-run budgets, timeouts, and spawn depth limits bound the cost of any single runaway run, and the kill switch halts one on demand. There is not yet a per-tenant global governor, so a trigger storm, for example a mail loop feeding an inbound-document trigger or a misconfigured schedule, could start many individually-within-budget runs whose aggregate cost is

painful. Metering makes the aggregate visible quickly, and the exposure is financial and operational rather than a break in confidentiality or integrity. A per-tenant rate and spend governor is the fix and is planned.

**Communications adapters are not shipped.** Any workflow whose last step sends something outward closes that loop with a person today. That is a scoping fact for a pilot, not a surprise for month three.

**Graduation thresholds have no long operating history.** The mechanism is built and the defaults above are reasoned rather than tuned against years of production data. Expect to adjust them per queue, and expect the first few graduations to be conservative.

The threat model for this machinery is published rather than summarized. Approval fatigue and evidence gaming is a tracked threat with a Monitored residual, meaning real exposure remains and named detection surfaces watch it. We would rather hand you that document than have you find the same conclusion on your own.

## Further reading

*Governing AI access to systems of record* covers the security architecture underneath this: the invariants, identity, trust boundaries, the audit record, and the full accepted-risk list. *The Omnafy platform architecture* covers the gateway, the planes, and the deployment split in engineering detail. *HIPAA and the AI governance layer* covers the compliance posture and the shared-responsibility split.