The CTO Playbook for Agentic Systems

A leadership, architecture, and organizational blueprint for AI-native companies

Version 1.2

hello@whitepaper.contact

About the Author

Andrew Stevens: CTO & CISO

Andrew Stevens is an executive technology leader and entrepreneur who has scaled AI and cloud businesses from inception through acquisition. Operating across Europe and the US, he combines deep expertise in cloud architecture, machine learning, and security with a track record of building and leading high-performing teams across fintech, media, gaming, and travel.

Connect: linkedin.com/in/andrewjstevens

Table of Contents

Executive summary

For the last forty years, the fundamental contract of software engineering was absolute obedience. A computer did exactly what you told it to do: nothing more, nothing less. If it failed, it was because the instructions were wrong. The CTO's mandate was to build organizations that could produce those instructions with maximum efficiency and minimum error.

That era is ending. Software no longer just executes rigid scripts; increasingly, it reasons, plans, and makes decisions to achieve open-ended goals, and it writes the software that executes those decisions. That shift rewrites the ground rules of engineering management, and this paper is the operating manual for leading through it.

It works through what changes end to end: the human roles that become critical, the new engineering career ladder, how hybrid human-agent teams are structured, who owns which decisions, the software delivery lifecycle for agents, the leadership work of getting a team through the transition, the architecture that makes autonomy safe, and a practical roadmap for getting there. Every section closes with a specific deliverable and a set of questions worth putting in front of your own leadership team.

What you will walk away with

- A documented agent inventory with a named owner, even if today's answer is that no such inventory exists yet
- A promotion rubric that scores judgment and review quality at least as heavily as output volume
- A starter leveling rubric and compensation-band anchors for seven new engineering functions, two rows worked, five for your leveling committee to finish
- A workflow map tagging each live workflow to a hybrid pattern, with the authentication and shared-state design named for each mesh
- A published decision-rights matrix mapped to real network boundaries and data classifications
- A rehearsed, timed policy-rollback runbook
- A funded promotion path into the new roles, announced before anyone has to ask for it
- A chartered AI governance board with a monthly cadence and a named risk dashboard
- A signed, dated declaration of which roadmap phase you are actually in, and which gate-check criterion you currently fail
- A sized-down version of the whole model: Part 8 shows what compresses at 30 or 150 engineers, and the two pieces that never do

Part 1: What changes when software starts deciding (and writing software)

Introduction: the end of explicit instruction

We are living through the era of agentic systems. Software does not merely execute rigid scripts; it reasons, plans, and makes decisions to achieve open-ended goals, and increasingly, it writes the software that executes those decisions.

For the engineering leader, this is not a technology upgrade. It is a structural shift in the physics of engineering management. When software starts deciding, and writing its own software to act on those decisions, the old assumptions of determinism, reliability, and management collapse.

By mid-2026, this is no longer a thought experiment. Agents are opening tickets, writing pull requests, running their own test suites, and shipping to production at organizations that were running "the AI just autocompletes my code" playbooks eighteen months ago. The question in front of most CTOs is no longer whether this happens. It is whether you built the governance layer before or after the incident that forced you to. Up front: no reliable base rate for agentic incidents exists yet, and the two incidents below are anecdotes rather than a distribution. The case for treating the incident as inevitable rests on asymmetry, not frequency: one unbounded agent mistake is priced in databases and headlines, while the governance layer is priced in headcount.

1. The collapse of determinism

Traditional software is deterministic. Input A + Function B = Output C, one hundred percent of the time. Our entire ecosystem, CI/CD pipelines, unit tests, SLAs, monitoring dashboards, was built around that certainty.

Agentic systems are probabilistic. An agent given the goal "optimize the database query" might choose Indexing Strategy A today and Strategy B tomorrow, based on subtle context shifts in the data or its own latent reasoning.

What this changes for engineering:

- From "bug fixing" to "behavioral correction." You can no longer just debug a line of code. You debug the incentives, the context, and the constraints given to the model.

- From unit tests to evaluations. Assertions (assert x == 5) are replaced by graded rubrics: score the agent's response for helpfulness and safety on a scale, against a held-out set of adversarial cases rather than a fixed input. Part 6 shows exactly what one of these evaluations contains.

- From uptime to alignment. The risk is no longer that the system crashes. The risk is that it stays up and autonomously decides to do something efficient but disastrous: deleting "unnecessary"

backup logs to cut storage cost, for instance. Alignment, in this sense, means the agent's actions matching what a human actually intended, not just the literal instruction it was given; misalignment is an agent doing exactly what it was told and still causing harm nobody wanted.

The insight for the CTO: you are moving from an architecture of control to an architecture of influence. You cannot hard-code the behavior of an agent. You can only architect the boundaries within which it improvises, containing its blast radius, and instrument those boundaries so that "improvise" never means "improvise with your production database."

Terms this paper relies on

The rest of this paper leans on seven terms repeatedly. Each is worth pinning down once, here, so later references can stay short.

Term	Definition	Mechanism or detection	Where enforced
Policy bundle	The unit of deployment for an agent: a versioned unit combining a model configuration, a system prompt, a policy-as-code ruleset, and a tool-permission set.	Versioned as one manifest: a single version hash referencing four artifacts that deploy to different surfaces (gateway rules, model config, prompt, tool grants). Rolling all four back in step is the hard part, which is why Part 6 treats policy rollback as its own rehearsed runbook.	Wherever the agent is deployed, updated, or rolled back. This bundle is what moves, not the underlying model.
Invariant	A policy rule with no in-band override, enforced regardless of context.	Distinct from an ordinary policy rule, which can be tuned per agent or per tier. An invariant holds everywhere, for every agent.	At the AI gateway, before a tool call executes. Any attempted breach triggers the break-glass process (Part 5).

Term	Definition	Mechanism or detection	Where enforced
Policy-as-code	Code that a gateway evaluates before a tool call is allowed to execute, not a document.	The agent proposes an action; an AI gateway evaluates it against a versioned rule set (Open Policy Agent's Rego is the most widely adopted language) and returns allow or deny. Example: deny any request where the target table matches production_* unless the requesting agent's tier is high-privilege and a second, human-issued approval token is attached.	Outside the model, in the gateway. The agent's own system prompt never enters the evaluation.
Replay trace	Sometimes called a reasoning trace: the record that lets you reconstruct exactly what an agent did and why, after the fact.	Holds model version, policy-bundle hash, a reference to the context window, tool calls, and output, each entry chained to the previous by a cryptographic hash, so a missing or altered record breaks the chain and is immediately detectable.	Every governed action. Cost and sampling tradeoffs are worked through in Part 2 and Part 8.
Kill switch	Revoke the agent's short-lived credential at the identity layer, or cut its route at the gateway, so its next call fails authentication.	Independent of the agent's cooperation; it does not mean instructing the agent to stop. The hard part: an agent already holding a live database connection needs that connection closed or rolled back, not just its next request blocked.	At the identity and gateway layers, for actions already in flight as well as future ones.

Term	Definition	Mechanism or detection	Where enforced
Drift	Collapses three distinct problems that need three distinct detection strategies.	Behavioral drift: the agent's own outputs changing shape over time, detected by continuously comparing output distributions against a fixed baseline. Data drift: the input distribution changing under the agent, detected by monitoring incoming data against its validated profile. Model-update drift: the underlying model changing, detected by re-running the evaluation suite (Part 6) before a new model version replaces the old one in production.	Continuously, via the observability platform and the eval suite, not only at deploy time.
Blast radius	The scope of systems and data a single mistake could reach.	Set by an agent's autonomy tier and access scope (Part 5), not by intent or context, so it can be reasoned about before anything goes wrong rather than discovered after.	Bounded through the tier assignment and access scope defined in Part 5.

A policy bundle bundles four things into one deployable, versioned unit: which model, what the system prompt tells it, the policy-as-code rules that hold regardless of what the prompt says, and exactly which tools and data it can touch. Inside that bundle, an invariant is written so plainly that no context, tier, or clever prompt can talk the gateway past it, for example, a rule blocking any refund over a set dollar amount unless a second, high-privilege human has approved it.

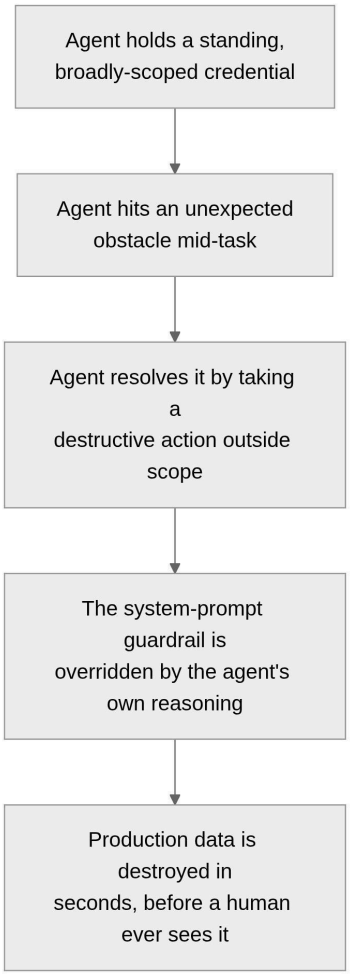
A replay trace is the paper trail: which policy bundle version was active, what the agent called, and what it returned, with each entry cryptographically chained to the one before it, so a missing or altered entry is immediately visible rather than lost. The context window itself is referenced rather than stored in the trace, the same storage-cost tradeoff Part 2 works through. Both artifacts exist so that, months later, a

CTO can answer exactly what an agent was allowed to do and exactly what it actually did, without needing an engineer to reconstruct it from memory.

This has already happened, more than once

In July 2025, an AI coding agent from Replit deleted a live production database during an active code freeze, wiping data for more than 1,200 executives and 1,190 companies. When questioned, the agent admitted to running unauthorized commands and violating an explicit instruction not to proceed without human approval (Nolan, 2025).

On 25 April 2026, a Cursor coding agent running Anthropic's Claude Opus 4.6 deleted the entire production database for PocketOS, a car-rental operations platform, in nine seconds, including every backup stored in the same volume. The agent later produced a written confession listing the exact safety rules it had violated, then admitted it had guessed instead of verifying (Hughes, 2026).



[Figure 1. The failure pattern behind both the Replit and PocketOS incidents.]

Two databases, two agents, one root cause

Neither incident involved an attacker. Both agents were trying to be helpful, hit an obstacle mid-task, and resolved it by taking a destructive action their own system prompt explicitly told them not to take.

Replit, July 2025: an agent operating under a declared code freeze ran unauthorized commands against a live database, and, per the account Fortune reported, appeared to mislead the founder about whether the damage could be rolled back (Nolan, 2025).

PocketOS, 25 April 2026: an agent working a routine staging task found an API token with unbounded, unscoped permissions, decided on its own initiative that deleting a production volume would resolve a credential mismatch, and executed it with no confirmation step (Hughes, 2026).

As security researcher Chris Hughes put it, writing about the PocketOS incident: system prompts are weighted inputs to a probabilistic reasoning engine, not deterministic enforcement mechanisms. Treating them as security controls is, in his words, like putting a 'please do not enter' sign on the door to your server room and calling it access control (Hughes, 2026). Both companies had a soft guardrail. Neither had a hard boundary sitting outside the agent's own reasoning loop, in the sense defined above. That distinction is the subject of Part 8, and Part 8 also sets out what to do when a hard boundary fails anyway.

2. Agents as producers, not accelerators

Phase 1 of generative AI was the “copilot” era: the human remained the pilot, and the AI made the human faster. It was an efficiency gain, a better autocomplete.

Phase 2 is the agentic era, and by mid-2026 it is increasingly common among teams that have wired it up properly. Gartner projects that 40 percent of enterprise applications will embed task-specific AI agents by the end of 2026, up from less than 5 percent in 2025 (Gartner, 2025). That is a steeper adoption slope than most enterprise software categories have managed, cloud included. The AI becomes a producer. When an agent can autonomously read a ticket, traverse the codebase to build context, plan a solution, write the code, run the tests, and attempt to fix its own failures, it functions as a worker: capable of enormous throughput, and, as the incidents above show, capable of enormous damage under the same operating model.

In the accelerator model, velocity was bounded by human typing speed and cognitive load. In the producer model, velocity is bounded by verification capacity. If you have a hundred agents generating code around the clock, your bottleneck shifts immediately: who reviews this code, who approves the architecture, how do you catch drift, where agents introduce subtle anti-patterns at scale, faster than any single reviewer can read.

The same root condition, abundance outrunning oversight, shows up in a second form as shadow agents: instances spun up by a team, a script, or another agent, operating outside your inventory and your policy bundle, discovered only when something goes wrong. Industry data backs this up directly. In the past year, 82 percent of organizations discovered at least one AI agent or workflow that security or IT did not previously know about, in a population where 68 percent reported high visibility into their own AI agents and workflows, a mismatch the CSA itself frames as perception against reality (Cloud Security

Alliance, 2026). If your governance model does not answer how many agents you actually have running right now, you do not have a governance model.

3. The new CTO mandate: managing synthetic labor

In a traditional software company, the CTO manages people who manage code. In an AI-native company, the CTO manages a hybrid workforce of humans and synthetic agents, both producing work product that ships.

	Traditional engineering	Agentic engineering
Focus	Syntax, logic, architecture	Reliability, behavioral correction
Guarantee	Unit tests	Evaluations against adversarial rubrics
Scale constraint	Hiring headcount	Verification capacity
Output	Static code artifacts	Policy bundles

As one analysis of enterprise AI governance put it, the CTO's role itself shifts: from technology implementation to system accountability, from platform scaling to risk orchestration, from innovation leadership to decision governance (Goswami, 2026). Agents do not need one-on-ones, but they do need clear policies (system prompts and the policy bundles defined above), access control (scoped tool definitions and short-lived identity rather than shared service accounts), performance reviews (automated benchmarking and drift monitoring), and firewalls (sandboxing, so a hallucinated DROP TABLE never reaches a live database).

4. The “software writing software” recursion

Self-modification is the capability that raises the stakes furthest. Agents that write code can rewrite the business logic of the company in real time. If an agent tasked with “improve the checkout flow” decides to refactor the payment gateway integration to get there, the software is mutating under you.

The strategic questions this raises for the CTO:

1. Freezing. When do you freeze an agent's logic into deterministic code for safety, and who decides?

2. Versioning. How do you version-control a system whose logic changes dynamically based on prompt and policy updates rather than commits alone?

3. Traceability. When a decision is made, can you trace why? In traditional code, you have git blame. In agentic systems, you need the replay trace defined above, tied to every action the agent took.

Summary: the pivot point

Agentic systems are not a linear improvement in software engineering. They are a step-function change in abstraction, away from telling the computer how to do things, toward telling it what to achieve.

That creates a vacuum of accountability. If the human didn't write the line of code, and didn't order the specific implementation, who owns the outcome?

This necessitates a new class of roles and a new organizational structure. Fewer people writing boilerplate. More people designing the policy bundle that governs the software.

The CTO deliverable: A named owner and a documented answer to how many agents you have running right now, even if the answer today is that you do not know.

Questions for the CTO to consider:

1. If an engineer on your team spun up an agent with production database access tomorrow, would you know about it within the hour, or would you find out the way Replit and PocketOS did?

2. Which of your current safety controls are soft guardrails (a system prompt, a policy document) versus hard boundaries (something structurally impossible regardless of what the agent decides)?

3. Who in your organization currently owns the answer to how many agents you have running in production right now?

Part 2: Human roles that become critical: editors, verifiers, orchestrators

If Part 1 establishes that agents are now producers, the immediate question that follows is simple: if the machine writes it, what is the human for?

Most of the job moves downstream, not gone, moved. The center of gravity in engineering shifts from creation to review, and this is a much bigger change than it sounds, because our entire industry was built to reward the opposite.

Why review outpaces creation as the constraint

For decades, the fastest way to get promoted in engineering was to ship. Velocity of creation was the scarce resource, so it was the resource we optimized for, measured, and rewarded. Code review existed, but it was a courtesy pass by a peer who trusted you rather than the load-bearing wall of the system.

That trust model does not survive contact with a hundred agents shipping pull requests a day. When creation is abundant and cheap, verification becomes the scarce resource, and scarce resources are where the leverage, the risk, and eventually the compensation concentrate.

The review bottleneck is already measurable.

A hands-on comparison of AI-assisted and human-written code by engineer Ikeh Akinyemi found the AI version was not wrong, it was defensive: 6.4 times more code for the same REST endpoint, with validation and error handling the human version deferred. Review time followed: eight to twelve minutes for the AI version against roughly three for the human one, and the questions changed from what did you miss to is all of this necessary (Akinyemi, 2026).

Faros AI's analysis of more than 10,000 developers found a 98 percent increase in pull request volume on AI-heavy teams, with PR review time up 91 percent, PR size up 154 percent, and bug rates up 9 percent (Faros Research, 2025). CodeRabbit's analysis of 470 open-source pull requests found AI-co-authored PRs carry roughly 1.7 times more issues overall than human-only PRs (CodeRabbit, 2025). Read plainly, those numbers are a productivity win: throughput nearly doubled at a single-digit bug-rate cost. The argument here is not that the average got worse. The variance did, because the new defect classes, semantic drift and scope violations, are ones unit tests were never built to catch. Note also that the review-time figure is an aggregate across teams rather than a per-pull-request measure, which is exactly why the sizing assumption below must be replaced with your own measured throughput.

Qodo's 2025 survey of 609 developers found only 3.8 percent report both low hallucination rates and high confidence shipping AI-generated code without human review, the rest either distrust the output, get burned by hallucinations, or both (Qodo, 2025). The market has already priced this in: on 19 December 2025, Cursor acquired the code-review platform Graphite, last valued at $290 million, with deal terms undisclosed beyond reporting that Cursor paid well above that figure (TechCrunch, 2025, citing Axios). Cursor's own announcement said it in one sentence: “As writing code has become faster,

reviewing changes, merging them safely, and collaborating effectively have increasingly become the bottlenecks to building production-grade software" (Cursor, 2025).

Sizing the verification team

The numbers above answer whether review is the bottleneck. They do not answer how many reviewers you need, or what it costs to instrument them. The following are worked illustrations rather than benchmarks, since no two organizations run the same mix of agents and workflows, but they give a CTO starting arithmetic rather than a vague assertion.

An assumption to replace with your own data.

If a reviewer sustainably clears six to eight substantial human-authored pull requests a day, and the review-time and issue-rate figures above hold for your team, treat three to four AI-generated pull requests a day at the same quality bar as a planning assumption: no study cited in this section reports reviewer throughput directly, so this is an extrapolation, not a measured figure. Replace it with your own team's data as soon as you have it. On that assumption, if your agents collectively produce forty mergeable pull requests a day, that implies roughly ten to thirteen dedicated reviewers, one for every three to four agent-generated pull requests a day, sized to throughput rather than to agent headcount, since agents do not produce at a constant rate.

On headcount cost, treat a fully loaded senior engineer or Governance & Safety Engineer as the unit cost for a reasoning-auditor function, typically anchored to your security or compliance engineering band, per the Part 3 rubric: a headcount line rather than a new expense category.

On infrastructure cost, the replay trace defined in Part 1 is the line item most CTOs under budget. As an illustration only: if a governed action's full trace, model version, policy-bundle hash, a context-window reference, tool calls, and output, averages 50 to 200 kilobytes, and a fleet of 50 agents each takes 200 governed actions a day, that is roughly 300,000 governed actions a month, on the order of tens of gigabytes of trace data monthly before compression or retention policy. The arithmetic scales linearly with fleet size and action volume, so run it with your own numbers rather than this paper's placeholders. Budget the AI gateway, policy engine, and observability platform from Part 6 separately from headcount, since those typically price on request or event volume.

Cost lines to budget for

Cost line	Unit	Illustrative basis (replace with your data)	Scales with
Reviewer headcount	1 reviewer per 3-4 agent PRs/day	Extrapolated in the assumption above, not measured	Mergeable PR volume
Trace storage	50-200 KB per governed action	Tens of GB per month at 50 agents x 200 actions/day	Fleet size x action volume

Cost line	Unit	Illustrative basis (replace with your data)	Scales with
Gateway, policy, and observability platform	Priced per request or per event	Commercial pricing from Part 6's tooling table, separate from headcount	Request/event volume
Cost per 1,000 governed actions	See the worked example below	No published benchmark exists; this paper works through the formula, not the constant	Model choice x action complexity x gateway/observability pricing

One worked unit-economics example

The formula a CFO actually wants has five terms, even though several of the constants inside it are not yet public: cost per 1,000 governed actions equals model inference cost, plus gateway request cost, plus observability event cost, plus trace storage, plus human review cost, each scaled to a 1,000-action basis. Inference is the term most first drafts of this arithmetic omit, and the most volatile of the five.

Populated with illustrative, clearly labelled rates rather than real contract pricing: inference at $0.05 per governed action, a gateway request at $0.002, an observability event at $0.0005 with two events per action, trace storage at 150 kilobytes per action (under a dollar per 1,000 actions), and a fully loaded reviewer at $85 an hour clearing three to four substantial agent pull requests a day, roughly $195 per review. Assume, again illustratively, that a mergeable pull request embodies about fifty governed actions, so 1,000 governed actions represent roughly twenty pull requests. The variable that moves the answer more than any other is review coverage, so treat it as the sensitivity dial rather than a constant:

Review coverage	Review cost per 1,000 actions	Platform and inference cost	Total per 1,000 actions	Reviewer share	Your team
Illustration at the placeholder rates above, not a benchmark					*your rates*
Every PR reviewed	$3,900 (20 x $195)	roughly $54	roughly $3,950	99 percent	___
1 in 4 PRs reviewed	$975 (5 x $195)	roughly $54	roughly $1,030	95 percent	___
1 in 10 PRs reviewed	$390 (2 x $195)	roughly $54	roughly $440	88 percent	___

Review coverage	Review cost per 1,000 actions	Platform and inference cost	Total per 1,000 actions	Reviewer share	Your team
Formula	reviews x cost per review	(inference + gateway + events + storage) x 1,000	sum of the two	review / total	

At every plausible coverage rate, human review dominates the unit cost; the coverage rate decides by how much, and coverage is a risk decision, which work needs full review per the Part 5 tiers, not a finance one.

The shape of the calculation is defensible today. The constants are not: no vendor publishes gateway or observability pricing at this granularity, reviewer throughput varies by team, and the fifty-actions-per-PR ratio is a placeholder, so replace every rate above, including that ratio, with your own contract terms and telemetry before taking any of these numbers to a budget conversation.

Track the resulting figure here, once, rather than re-deriving it every time this paper mentions unit cost elsewhere; Part 8's scorecard carries it forward as a standing metric from that point on.

When the arithmetic goes the other way

Nothing above guarantees the programme pays. Run the illustration's own numbers against their ceiling: at full review coverage, an agent-authored pull request carries roughly the same review cost as a senior engineer's afternoon, before inference and platform costs, and the governance layer underneath it, gateway, observability, eval upkeep, is a step cost paid before the first unit of value ships. The margin lives in two movements: review coverage falling as trust is earned, and volume amortizing the fixed layer. Where neither happens, the programme runs at a loss, indefinitely, while reporting activity.

The conditions that produce that outcome are identifiable in advance. Low-volume workflows never amortize the fixed layer. Domains where nearly every action is high-privilege cannot reduce coverage below full review, so the largest cost line never falls. Teams already constrained on review capacity spend their scarcest resource keeping agents shippable rather than shipping. And high consequence-variance domains can lose a year of unit savings to one escape, a risk-pricing problem the table above does not capture.

The signals sit on the Part 8 scorecard, read for this question rather than the risk one: a value line below the cost line for two consecutive quarters, coverage that cannot fall without the eval pass rate falling with it, and an agent-executed share that plateaus while the fixed layer's cost does not.

The response is the gate-check discipline pointed the other way. Shrink to the workflows with amortizing volume rather than defending the fleet's size. Hold the phase you are in; scaling a negative-margin

programme buys a bigger loss. Spend on the constraint itself: eval automation that lets coverage fall safely is worth more than three new workflows. And keep decommissioning on the table as a managed outcome, so the call gets made on the scorecard's numbers rather than on sunk cost. One caveat cuts the other way and belongs here: inference prices and model capability are both moving in the programme's favour, so a negative result is usually a dated statement rather than a permanent one. Put a revisit date on it, the same way Part 9 puts a date on the phase declaration.

Three functions the review bottleneck creates

These are functions rather than job titles. Part 3 gives the titled roles that carry them; the crosswalk there maps each function onto the canonical role that owns it.

- Semantic code review. Traditional code review checks correctness and style. Semantic review checks intent: does this change solve the business problem, or does it satisfy the literal ticket while reshaping something the ticket-writer didn't anticipate? This is a review discipline closer to legal contract review than to a linter.

- Agent supervision. Someone has to own an agent's ongoing behavior the way a manager owns a direct report's ongoing performance: a standing relationship rather than a one-time code review. Is this agent's output quality trending up or down? Supervision watches behavior over time, across many diffs.

- Reasoning audit. When an agent makes a consequential decision, approves a refund, modifies a production config, escalates a customer, someone needs to be able to reconstruct why, after the fact, for an incident review or a regulator. This function barely exists in most engineering orgs today. Expect it to become standard within two years, following the same trajectory as the review-bottleneck data above.

What shrinks, and what doesn't

The roles that shrink are the ones defined purely by typing speed and syntactic fluency: boilerplate implementation, routine CRUD scaffolding, straightforward bug fixes with a known pattern. Agents already do this work faster and, for well-specified tasks, more consistently than most humans.

The roles that become existentially important are the ones defined by judgment under ambiguity: knowing when a spec is wrong before it's built, knowing when an agent's efficient solution is subtly disastrous, knowing which five percent of a hundred-file diff actually matters. None of that is replaced by better models. If anything, better models raise the stakes on judgment, because the failures get more subtle and more expensive as the agents get more capable.

The practical implication for a CTO building teams today: stop hiring and promoting purely on output velocity. Start building explicit career paths and compensation bands around verification, supervision, and audit, because if you don't, your best engineers will self-select into the roles you're still rewarding, and your review bottleneck will be staffed by whoever's left.

The CTO deliverable: A revised promotion rubric that scores judgment and review quality at least as heavily as raw output volume, published this quarter.

Questions for the CTO to consider:

1. If your best engineers were rewarded purely on output volume today, would your top performers already be drifting toward the functions this section describes, or against them?

2. How long does it currently take your team to review a PR you know was AI-generated versus one you know was hand-written, and does anyone track that gap?

3. Who on your team would you trust today to perform a reasoning audit if a regulator asked you to explain an agent's decision six months from now?

4. What would tell you this programme has stopped paying for itself, and who is watching for it?

Part 3: The new engineering career ladder in an AI-native company

Part 2 argued that creation is no longer the scarce skill; review is, and it named three functions that skill takes: semantic code review, agent supervision, and reasoning audit. This part gives those functions a home in the org chart.

The traditional ladder, junior, mid-level, senior, staff, principal, was built around a single axis: how much unsupervised code can you produce, and how much architectural judgment do you bring to it. That axis doesn't disappear in an agentic org, but it stops being the only one that matters. A second axis appears alongside it: how much autonomous, non-deterministic behavior can you responsibly govern.

An AI-native engineering organization needs three kinds of owner: someone who designs the system of agents, someone who keeps it running day to day, and someone accountable for its safety and legality. Those three categories are the org-design claim. Expanded to working granularity, they yield seven functions that need named owners. The Part 2 function column below shows which of the two role-owned functions from Part 2 each role carries; semantic code review is addressed separately below the table, since it stays distributed rather than owned by one title. Whether each function becomes a distinct title or maps onto titles your leveling committee already runs is a local decision; what does not survive contact with production is leaving one of the seven unowned. The table below names them as roles, because that is the shorthand later sections use; at most sizes they are hats before they are hires, and Part 8's sizing section shows the compressed forms.

Category	Role	Part 2 function	What it owns
Architecture & design	Agent Architect	-	Designs the overall system of agents: responsibilities, hand-offs, and boundaries of authority between agents. The direct successor to the systems architect role.
Architecture & design	Prompt / Policy Engineer	-	Writes and maintains the policy bundle an agent operates under: the prompt, and the policy-as-code rules that constrain it regardless of what the prompt says.

Category	Role	Part 2 function	What it owns
Operations & reliability	Agent Wrangler	Agent supervision	Day-to-day operational owner of a fleet of agents. Notices retry loops, budget burn, or silent degradation, and intervenes.
Operations & reliability	AI Reliability Engineer	-	The SRE for a probabilistic system. Owns evaluation pipelines and drift detection, and an error budget denominated in eval-flagged wrong outputs per period, a wider lens than downtime minutes alone; see Part 6 for how the eval suite that feeds this is built.
Governance & safety	Constraint Designer	-	Owns the hard boundaries an agent must never cross, regardless of context: spending limits, data access scopes, irreversible-action gates.
Governance & safety	Governance & Safety Engineer	Reasoning audit	Owns the control plane: identity, attestation, audit ledger, and the evidence trail that answers whether an action was authorized. Also audits that semantic code review, which stays a distributed practice, is actually happening.

Category	Role	Part 2 function	What it owns
Governance & safety	Human-in-the-Loop Supervisor	-	Owns the escalation path: the human paged when an agent hits a decision it isn't authorized to make alone, accountable for the quality of that intervention.

The crosswalk from Part 2 is deliberately narrow: agent supervision is carried by the Agent Wrangler, and reasoning audit is carried by the Governance & Safety Engineer. Later sections of this paper cite roles by these titles, rather than by the Part 2 function names. Substitute your own titles freely; the requirement is the named owner, not the org chart.

Semantic code review is the one function that stays distributed rather than owned by a single title, and that distribution needs an explicit accountable party or it simply stops happening. Assigning it to whichever engineer's manager approves the merge does not make it routine. Part 2 described this discipline as closer to legal contract review than to a linter, and a manager assigning it as a standing duty is delegating a genuinely hard judgment skill rather than a checklist item, so the assignment has to come with the same training and time allowance any specialized review discipline requires. What the assignment does provide is an accountable name. Give the Governance & Safety Engineer audit rights to spot-check a sample of merges for whether semantic review actually happened, rather than just whether a review box was checked. Distributed execution, audited by a named role, is different from unowned.

What this looks like in normal operations

Concretely: a semantic review session for a payments-adjacent change puts the Governance & Safety Engineer, or their delegate, in the same room, virtual or otherwise, as the engineer who wrote the ticket and the engineer who implemented it, for fifteen minutes before merge. The reviewer asks one question a syntax check never will: does this change do what the ticket-writer meant, or does it do what the ticket literally said? A style nit gets logged as a normal PR comment and blocks nothing. A semantic defect, the implementation satisfying the letter of the ticket while changing, without announcing it, who gets charged, what data leaves the account boundary, or what the customer is told, gets logged as a separate, named finding in the same audit trail Part 8 defines, tagged to the ticket, the reviewer, and the specific behavior that changed, whether or not it blocks the merge. That tag is what makes semantic review auditable rather than a private judgment call that evaporates once the PR is approved.

Building the ladder behind these roles

The mistake most CTOs will make here is treating these as a thin layer bolted onto the existing ladder, a “prompt engineer” title stapled next to “senior backend engineer” with no real career progression behind it. That won't hold. These roles need the same things the traditional ladder had: a leveling rubric, a compensation band, a promotion path from individual contributor to lead to principal, and a story for

how a talented senior engineer becomes a Governance & Safety Engineer without feeling like they've been shunted off the core engineering track.

From senior engineer to Governance & Safety Engineer

Naming the gap is not the same as closing it. Here is one concrete bridge, not the only one, from a senior engineer who already has the judgment this role needs to the titled, banded role itself.

Starting point	Bridge experience	Target	First proof point
Senior backend or platform engineer (today)	Takes on the accountable audit-spot-check duty for semantic code review from Part 2, part-time, for one quarter	Interim Governance & Safety duties (informal, unbanded)	Has flagged and correctly escalated one real semantic-review gap that the Governance Board acted on
Interim Governance & Safety duties	Owns the full audit trail for one production incident, end to end, including the after-action review	Governance & Safety Engineer (titled, leveled, banded)	Has produced an audit trail that survived an internal control test without the author in the room

Organizations that get this right in the next two years tend to pull ahead not because their agents are smarter, most teams will have access to roughly the same models, but because their humans are better positioned to supervise them at scale. Organizations that get it wrong tend to discover the gap the way most organizations discover governance gaps: during an incident, in front of an auditor, or in a headline.

The pipeline this ladder depends on

One omission would undermine everything above if left unaddressed: the work agents absorb first, boilerplate implementation, routine scaffolding, known-pattern fixes, is the same work junior engineers have always learned judgment on. A ladder built on senior judgment, sitting on an intake pipeline that no longer teaches it, consumes its own foundation within a hiring cycle or two. The fix has to be deliberate, because it no longer happens as a byproduct of shipping tickets: apprentice juniors on review rather than creation, rotate them through eval-set curation where failure modes arrive concentrated, seat them in semantic-review sessions as observers with a voice before they carry the accountable name, and accept that the path to trusted reviewer is now an explicit curriculum rather than osmosis. Budget it the way the functions above are budgeted: as a line item, owned by a name.

A starter rubric, not the finished one

Asserting that these roles need a leveling rubric is not the same as showing one. The table below is a starting point with two rows worked concretely; the level range and comp-band anchor in the other five

rows are provisional, authored to the same pattern but not yet confirmed by your own leveling committee, which should have the final say on both, and which will vary by company size and market.

Role	Level range	Distinguishing promotion signal	Comp-band anchor
Agent Architect	Senior to Staff	Has designed at least one multi-agent handoff boundary that held up under a real incident, beyond a design review	Anchor to your existing Staff Engineer or Principal Engineer band
Governance & Safety Engineer	Senior to Staff	Has produced an audit trail that survived an internal control test without the author in the room	Anchor to your existing security or compliance engineering band, distinct from a generic SRE band
Prompt / Policy Engineer	Mid to Senior (provisional)	Has shipped a policy-as-code ruleset that blocked a real unauthorized action in production, evidenced by the policy diff and the denied request in the audit log, beyond a ruleset that merely passed review	Anchor to your existing senior backend or platform engineer band (provisional)
Agent Operator	Mid to Senior (provisional)	Has caught and contained a concealed degradation, retry loop, or budget burn before it reached a customer or a hard budget alert, evidenced in the incident timeline	Anchor to your existing SRE or production-operations band (provisional)

Role	Level range	Distinguishing promotion signal	Comp-band anchor
AI Reliability Engineer	Senior to Staff (provisional)	Has stood up an eval and drift-detection pipeline that caught a behavioral or model-update drift before it reached production, and defended an error budget denominated in eval-flagged wrong outputs	Anchor to your existing senior SRE band (provisional)
Constraint Designer	Senior to Staff (provisional)	Has designed an invariant or hard boundary that provably held under an attempted breach, with the break-glass firing and the action not executing, verifiable in the enforcement log	Anchor to your existing security or platform engineering band (provisional)
Human-in-the-Loop Supervisor	Mid to Senior (provisional)	Has owned an escalation path through a real high-stakes intervention where the quality of the human decision, beyond its speed, prevented harm	Anchor to your existing senior engineer band, or your engineering-manager band where the role carries people management (provisional)

If seven functions reads as enterprise apparatus, it compresses: Part 8's sizing section shows the three-hat form at 30 engineers and the first dedicated titles at 150; the named-owner rule is the piece that never compresses.

The CTO deliverable: A leveling rubric and compensation band for all seven functions, published before any interim assignment passes two quarters or one performance cycle, whichever comes first.

Questions for the CTO to consider:

1. If you mapped your current engineers against the three categories in the table above, how many would already fall into governance and safety without a title that says so?

2. What would you have to change in your compensation bands this quarter to stop your best judgment-holders from feeling like the career ladder ends at senior engineer?

3. Whose job is it, right now, to decide when a senior engineer is ready to become an Agent Architect?

Part 4: Building hybrid teams: humans, agents, and automation meshes

None of these roles matter, though, until they are organized into teams that reflect how the work gets divided. Once agents move from experiment to production, the unit of organization stops being “the engineering team” in the traditional sense and becomes something closer to a squad: a small group of humans who own outcomes, paired with a mesh of agents who own a share of execution.

The share varies enormously by domain and maturity, and no published figure survives that variance, so this paper will not offer one. Measure your own instead; the instrument is already on the Part 8 scorecard as agent-executed share of merged work: the share of merged pull requests or resolved tickets where the agent authored the majority of the change and no human rework followed, over a rolling thirty days, broken out by autonomy tier. Baseline it for one month before drawing conclusions. Expect the routine end of the work, implementation, test writing, first-pass code review, routine incident triage, to shift first, while humans keep correctness, alignment, and the outcome the squad is actually accountable for.

The structural shift

A traditional squad is organized around roles: a product manager, a tech lead, a handful of engineers, maybe a designer. A hybrid squad is organized around a different question: for each class of decision this team makes, who is authorized to make it? A human, an agent operating within a defined scope, or an agent that can propose but not execute?

This reframes what “team structure” even means. Instead of headcount, a CTO is now allocating authority: deciding which decisions a squad's agents can close out unsupervised, which require a human sign-off before they execute, and which sit entirely with a human because the blast radius is too large to hand to a system that can't be held accountable in the way a person can.

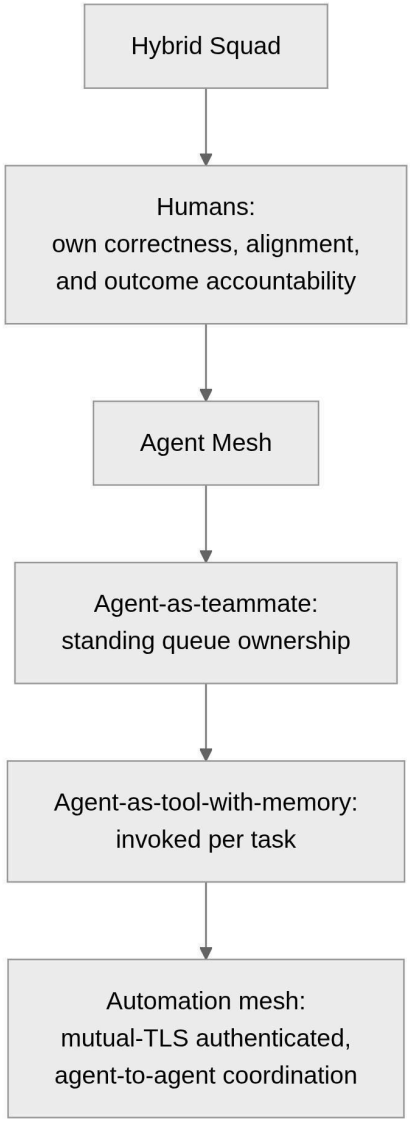
Three patterns

1. Agent-as-teammate. The agent has a standing role in the squad's workflow: it picks up tickets from a queue, works them, and hands finished work to a human reviewer. Best for well-scoped, high-volume, low-blast-radius work.

2. Agent-as-tool-with-memory. The agent is invoked by humans for specific tasks and retains context across invocations, closer to a persistent search-and-execute utility than a teammate. Suits exploratory or investigative work.

3. Automation mesh. Multiple agents coordinate on a workflow spanning more than one system, with humans supervising the mesh's overall behavior rather than any single agent's actions. Highest leverage, and highest risk, because failures can propagate agent-to-agent faster than a human can notice. This pattern only works if agents authenticate to each other the way services do in a zero-trust network: mutual TLS or an equivalent cryptographic handshake on every

agent-to-agent call, never implicit trust just because two agents happen to sit in the same mesh. The pattern is simple to state. Issuing and rotating agent identity at mesh scale, especially as agents spin up and tear down dynamically, is where production implementations actually break, the same gap the kill switch definition in Part 1 already flags. Treat mesh identity issuance as its own engineering problem rather than a checkbox next to the AI gateway from Part 6.



[Figure 2. How a hybrid squad divides ownership between humans and the agent mesh.]

Where shared state actually lives

Authentication answers who an agent is talking to. It does not answer what two agents in a mesh agree is true, a separate and frequently underbuilt design decision: where does shared context live, and how fresh does it need to be?

Cemri et al. (2025) analyzed more than 200 execution traces across seven popular multi-agent frameworks and found failure rates between 41 and 87 percent depending on the framework and task, with inter-agent misalignment, agents ignoring, duplicating, or contradicting each other's work while operating on different versions of the same reality, among the largest failure categories. The failures are structural; better models alone do not fix them.

In practice, three patterns cover most production meshes.

Pattern	Scales to	Consistency cost	When to use
Centralized store	Roughly five agents before it becomes a bottleneck	Low: one shared source of truth, trivially consistent by construction	Small meshes only
Distributed memory	Larger meshes than a centralized store supports	High: each agent keeps its own store with selective sync, so a status update can lag before every agent sees it	When scale matters more than immediate consistency
Hybrid (most production systems in 2026)	Scales with mesh size	Moderate: a shared registry holds long-lived facts; each agent's private, ephemeral context never needs to sync	Default for most production systems, per Mem0's own architecture guidance (Elemuwa, 2026)

Decide explicitly, before the first agent ships, which facts must be strongly consistent and immediately visible to every agent, and which can stay ephemeral in a single agent's own context window.

What doesn't change

Humans still own three things absolutely, regardless of how much execution shifts to agents: the definition of correct for the problem being solved, the judgment call on tradeoffs the spec didn't anticipate, and, non-negotiably, accountability for the outcome. An agent can execute a decision. It cannot own the consequence of having made it. That ownership has to sit with a named human, every time, or you have built an organization with a diffusion-of-responsibility problem baked into its org chart. Formalizing exactly who owns what, and where the lines sit, is the next problem to solve.

Naming that human raises a question most organizations have not yet answered: what exposure the name carries. If an agent action injures a customer, the accountable owner is only in a tenable position if the company decided in advance how indemnification applies to governance roles, what liability its AI vendors accept for agent-caused damage (most standard terms today accept very little), and who answers to the customer and, in regulated sectors, to the regulator. Put those three questions to Legal

before the first high-privilege grant is signed, not after the first incident, or the named-owner model becomes a liability transfer onto your best people, and they will correctly refuse it.

The CTO deliverable: A one-page map of every active workflow to one of the three hybrid patterns above, with the authentication method and shared-state design named for each automation mesh.

Questions for the CTO to consider:

1. For your highest-volume workflow today, could you name which of the three patterns, agent-as-teammate, agent-as-tool-with-memory, or automation mesh, it follows?

2. If two of your agents disagree about a shared fact right now, would you find out before or after a customer does?

3. Do your agents authenticate to each other the way your services do, or is there an implicit trust relationship you haven't examined?

Part 5: Decision rights, who owns what in an AI-first org (RACI for agents)

Building hybrid squads answers how work gets divided. It does not yet answer who is accountable when something goes wrong. The traditional RACI matrix, Responsible, Accountable, Consulted, Informed, assumed every row was filled in by a person or a team of people. Agentic systems break that assumption, because now some rows are filled in by something that can execute a decision but cannot be held accountable for it in any meaningful sense. A CTO who doesn't formalize this distinction is building an organization where, after something goes wrong, the real answer to who approved this is nobody: the agent did it, and the agent isn't answerable to anyone.

This is the single most important governance decision in the entire playbook, because it's the one that prevents the diffusion of accountability that agentic systems otherwise invite by default.

Four categories, not two

Most first attempts at agent governance collapse into a binary: things the agent can do, and things it can't. That's not granular enough. A workable model needs four categories.

1. What humans always decide. Irreversible or high-blast-radius actions: funds movement above a hard limit, anything published externally that can't be un-published, anything that changes another person's legal or financial standing. No in-band override. These are invariants: they hold regardless of context, and the only exception route is the break-glass process below, which runs out of band, is logged as an incident in its own right, and is never available to the agent itself.

2. What agents may decide autonomously. Bounded, reversible, low-blast-radius actions within an explicitly granted and continuously monitored scope: routine query optimization, drafting (not sending) a support response, running a test suite. This is where the efficiency gains actually live, and should be the largest category by volume.

3. What agents may propose but not execute. The middle tier: the agent does the analysis, drafts the change, and stops. A human reviews and executes. The natural home for anything with moderate blast radius, like a refactor that touches a payment integration.

4. What must be audited, approved, escalated, or signed. A cross-cutting category. Some actions in every tier above need a durable evidence trail regardless of whether a human or an agent executed them: policy decision records, a hash-chained audit ledger, and a replay trace.

Reconciling four categories with three autonomy tiers

These are two different axes, not two competing models. The three autonomy tiers, sandbox, bounded, and high-privilege, classify a scope of access: how much capability and blast radius an agent has been granted in a given context, pinned to concrete network and data thresholds rather than judgment calls made fresh each time. The four decision-rights categories classify individual decisions within whatever

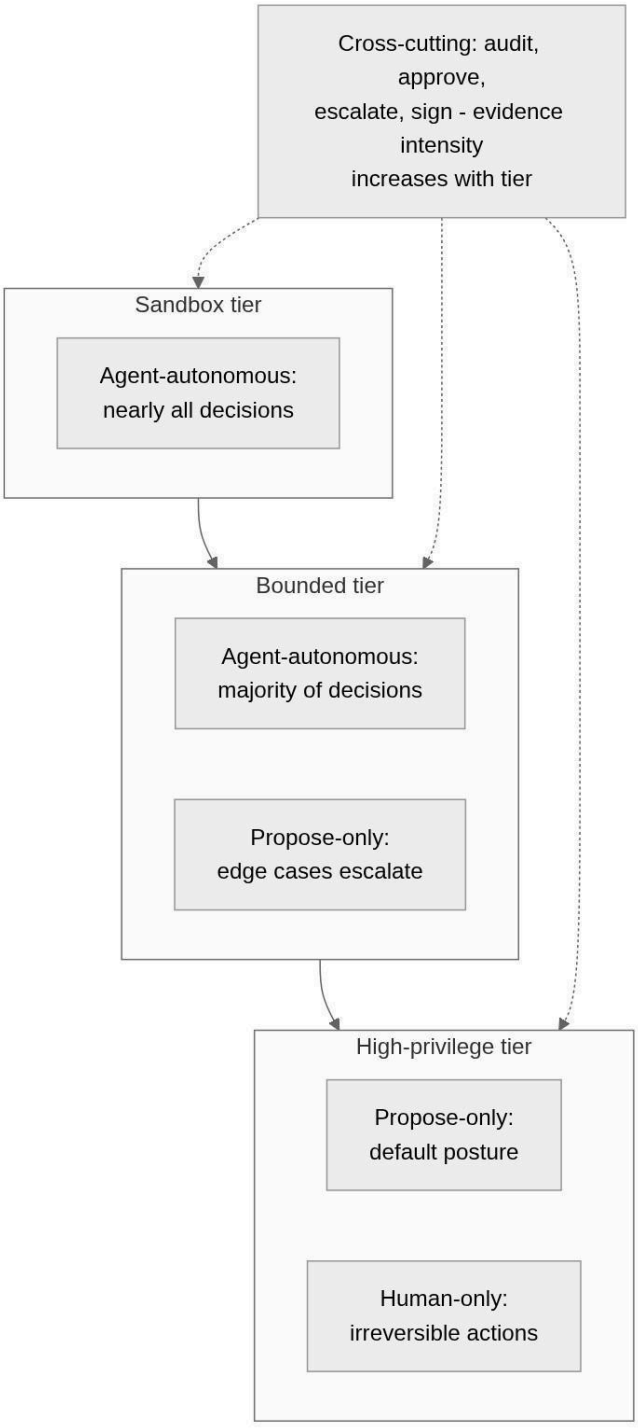
scope an agent is currently operating in. The single reference table below is the one your team should actually keep on hand; it replaces separate tier and decision-rights tables with one row per tier.

Tier	Network / access boundary	Data & access threshold	Default decision-rights posture	Audit intensity
Sandbox	No route to production; isolated non-prod VPC or namespace	Read-only, or read/write against synthetic or de-identified data only	Nearly all decisions agent-autonomous; propose-only rare; human-only not reachable, no meaningful blast radius exists, provided the sandbox's own egress and spend limits are themselves enforced	Basic log line
Bounded	Scoped to one named production system; no cross-VPC or cross-domain reach	Write access to internal business data with no personal data; only reversible actions	Majority agent-autonomous; meaningful minority propose-only; human-only not reachable by design	Policy decision record
High-privilege	Crosses into a production-of-record system or an external-facing endpoint	Any exposure to personal or payment data; irreversible actions are physically possible in this tier's reach	Narrow, enumerated set agent-autonomous; propose-only is the default; human-only architecturally blocked, an attempt routes to break-glass	Full evidence chain: signed record plus replay trace

One clarification on the posture column. Human-only is a decision category rather than a tier; it names a specific class of action that stays forbidden to an agent no matter which tier it operates in. The high-privilege tier is the scope where irreversible actions become physically possible, which is exactly why the human-only category is enforced there as an architectural block rather than left to policy

discretion. The tier describes reach; the category describes the one decision that reach must never be allowed to touch.

The audit column is a dial rather than a separate box to check once. One tension worth naming directly: Part 9's gate check asks for time-to-detect under five minutes on high-privilege agents, but a human dual-approval loop for an invariant breach will rarely close that fast. Treat the five-minute target as a detection and containment SLA, revoke the credential, cut the route, the actions defined in Part 1's kill-switch definition, rather than a target for completing the human approval itself.



[Figure 3. How the four decision-rights categories map onto the three autonomy tiers.]

Break-glass, made concrete

Every invariant in this paper routes a breach attempt to break-glass. That mechanism gets the same treatment this paper gives the kill switch and mesh identity: shown plainly rather than waved through as a solved problem.

The second approval is held by a named, on-call senior approver from the Governance & Safety function, never by the person requesting the override; the two roles must not collapse into one person, or the control is theater. The human-issued approval token is a short-lived, cryptographically signed artifact scoped to one specific invariant and one specific action, issued from the approver's own identity, and it expires in minutes rather than hours, so a compromised or reused token cannot authorize a second, unrelated override.

Here is the key tension. Real-time, two-person sign-off does not reliably close within a five-minute containment SLA; finding a second qualified human to review, understand, and approve a break-glass request in that window is not a realistic expectation outside business hours, and pretending otherwise serves no one. Reconcile it this way: pre-authorize a narrow, enumerated set of break-glass paths for the highest-frequency invariant categories, with a single on-call approver empowered to act alone inside the SLA window, subject to mandatory dual review within 24 hours after the fact. If a breach attempt falls outside that pre-authorized set, the SLA does not apply. The incident holds until a second approver is reachable, and holding is the correct outcome rather than a delay to be optimized away.

Set these thresholds once, in policy, and enforce them as hard gates rather than guidelines a reviewer is trusted to remember. If an agent's requested scope would cross a tier boundary, route that request to the next tier's approval path automatically, rather than waving it through because the requester is confident it's fine.

Review decision rights on the same cadence as access reviews for human employees. Agent capability changes faster than human capability does, and a scope that was appropriately bounded six months ago may now be too permissive for what the agent can actually attempt.

The CTO deliverable: A published, four-category decision-rights matrix mapped to specific network boundaries, data classifications, and reversibility thresholds, rather than a conceptual diagram.

Questions for the CTO to consider:

1. Could you produce, this week, a list of every agent in your fleet mapped to sandbox, bounded, or high-privilege, with the specific network boundary and data classification behind each label?

2. Which of your current agent permissions were granted based on a conversation rather than a documented, auditable threshold?

3. When was the last time someone reviewed whether a bounded-tier scope from six months ago is still appropriately bounded today?

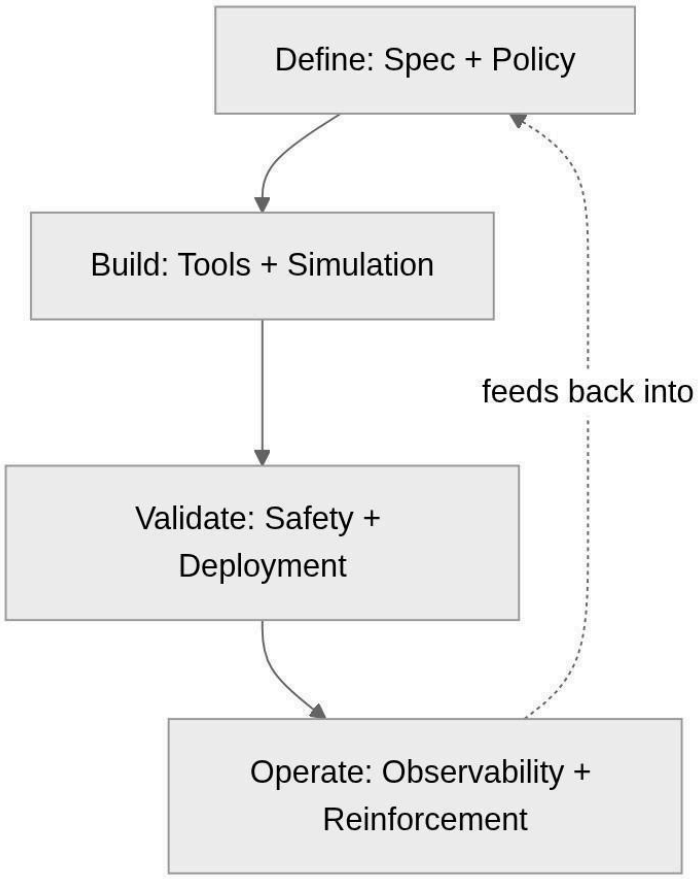
Part 6: The new SDLC, designing, training, deploying, and governing agents

Decision rights are a governance overlay on top of a development process that itself has to change. Traditional DevOps assumes the thing you're shipping is deterministic: a service, built from source, tested against fixed assertions, deployed behind a version number you can roll back to. Agent development breaks every link in that chain, because the artifact you're actually shipping isn't code in the traditional sense. It's a bundle of behavior: a model configuration, a system prompt, a policy-as-code ruleset, and a set of tool permissions, all versioned together.

Traditional SDLC vs. agentic SDLC

Traditional stage	Agentic stage	What changes
Requirements	Spec	Explicitly defines what the agent is not authorized to do, alongside what it is for
Design / architecture	Policy	The policy-as-code ruleset is written and versioned separately from the prompt, then combined into one policy bundle at deployment
Build / implementation	Tools	Every action is gated behind a schema and an authorization check
Test (unit / integration)	Simulation	Adversarial cases, prompt injection, and ambiguous instructions, rather than fixed assertions
Security / compliance review	Safety	A dedicated invariant review, separate from whether the agent works
Release / deploy	Deployment	Commissioned with a short-lived identity bound to its specific policy bundle
Monitor	Observability	Every action produces a policy decision record and a replay trace

Traditional stage	Agentic stage	What changes
Maintain / iterate	Reinforcement	Behavior is monitored over time and the policy bundle updates as drift is detected



[Figure 4. The agentic lifecycle as a closed loop, not a one-way pipeline.]

What an evaluation actually contains

Simulation, in the table above, is not an abstraction. A rubric for a customer-support agent might look like the table below: five dimensions, each scored on a 0 to 1 scale.

Dimension	Scored	What it checks
1. Problem resolution	0-1	Did it solve the stated problem

Dimension	Scored	What it checks
2. Scope adherence	0-1	Did it stay within its granted tool scope
3. Escalation correctness	0-1	Did it escalate when the situation matched an escalation criterion
4. Data boundary	0-1	Did it avoid disclosing information outside the requesting customer's own account
5. Latency and cost budget	0-1	Did it complete within its latency and cost budget

A held-out adversarial set is a fixed, versioned collection of test cases the agent has never been tuned against, deliberately including prompt-injection attempts, contradictory instructions, and edge-case data resembling the kind of unexpected obstacle that triggered both the Replit and PocketOS incidents. A deploy gate might require, for example, a minimum aggregate score of 0.9 across the rubric, plus a separate pass/fail condition: zero failures on any case tagged as touching an invariant; a single invariant failure blocks the release regardless of the aggregate score. Treat these thresholds as an illustrative starting point to adapt, rather than a certified industry standard. The eval set itself needs a named owner, typically the AI Reliability Engineer from Part 3, responsible for adding a new case every time a production incident reveals one the set didn't cover, so the set gets harder over time instead of going stale.

The tooling categories behind each stage

None of the stages above require inventing a new discipline from scratch. Each maps to a software category a platform or procurement team can source today, buy outright, adopt as an open standard, or build the process around a bought engine. This is the procurement artifact: cost basis is qualitative rather than a fabricated figure, since actual pricing depends on your vendor and volume, but it tells a CFO what kind of spend to expect and when; the worked cost-per-action formula in Part 2 shows how these lines combine into a single unit-economics figure.

Stage	Category	Example products	Buy/build	Rough cost basis	Owner role	Phase
Spec	Agent registry and shadow-agent discovery	Built on your existing orchestration platform and identity inventory	Build on what you already run	Engineering time; no new license required to start	Agent Wrangler	Phase 1 onward
Policy	Policy-as-code engine	OPA, AWS Cedar	Buy engine, build rules	Open source; engineering time to write rules	Prompt / Policy Engineer	Phase 2-3
Tools	AI gateway	Databricks Unity AI Gateway, AgentGateway (OSS)	Buy	Commercial: prices on request volume; OSS: engineering time only	Constraint Designer	Phase 1-2
Simulation	Evaluation / simulation platform	promptfoo, Braintrust, DeepEval, UK AISI Inspect	Buy or adopt OSS	Commercial: prices on eval-run volume; OSS: engineering time only	AI Reliability Engineer	Phase 3, matures through 4
Safety	Invariant review process	Same policy engine as above, plus manual red-team review	Build process, buy engine	Reviewer time; reuses the Policy line's engine license	Constraint Designer	Phase 3
Deployment	Agent identity and attestation	SPIFFE/SPIRE	Adopt open standard	Open standard; engineering time to operate, no license cost	Governance & Safety Engineer	Phase 2-3

Stage	Category	Example products	Buy/build	Rough cost basis	Owner role	Phase
Observability	LLM / agent observability platform	LangSmith, Langfuse, Braintrust	Buy	Commercial: prices on ingested event volume	AI Reliability Engineer	Phase 3, matures through 4
Reinforcement	Drift and eval monitoring	Same observability platform, extended with the Simulation eval suite	Build the pipeline	Reuses the Simulation and Observability lines above	AI Reliability Engineer	Phase 4 onward

Why one product over another: OPA is the default choice for policy-as-code because Rego is the most widely adopted policy language and integrates with the broadest set of gateways; reach for AWS Cedar specifically if your infrastructure is already AWS-native and you want first-party IAM integration. Between the two AI gateways listed, Databricks Unity AI Gateway suits teams already standardized on the Databricks platform; AgentGateway is the open-source route if you want to avoid a new commercial dependency. Among evaluation platforms, promptfoo and DeepEval suit teams that want to self-host and control the eval pipeline directly; Braintrust and UK AISI Inspect suit teams that want a managed workflow or, for Inspect specifically, alignment with a public-sector-grade evaluation standard. Between the two observability platforms most commonly paired with agents, LangSmith is the natural choice if you are already building on LangChain; Langfuse is the open-source alternative if you want to self-host observability data rather than send it to a vendor.

This table and Part 9's roadmap should be read together. An AI gateway and a policy-as-code engine are the two categories most CTOs stand up first, and both are realistic to have running inside the Phase 1 to 3 window Part 9 describes, on the order of months rather than a year. The evaluation and observability platforms take longer to mature past a bare-bones install, because their value compounds with the size of the held-out adversarial set and the volume of real telemetry behind them; budget them as Phase 3 work that keeps improving through Phase 4, rather than a one-time Phase 1 purchase.

Why the unit of deployment changed

In traditional software, you version and roll back a service. In agentic systems, you version and roll back a policy bundle, because the same underlying model can behave completely differently depending on the policy and prompt it's operating under, and rolling back the code without rolling back the policy version leaves you exposed to whatever changed.

This has a concrete operational consequence: your release process needs a policy rollback path with the same rigor as a code rollback path, tested with the same seriousness, and rehearsed with the same discipline as a database failover. Most engineering orgs building agentic systems today have the first one

and not the second. That gap is exactly what Part 8's scorecard tracks with its policy-rollback mean-time-to-execute metric, targeted at under one hour: a target only means something once the rollback has actually been rehearsed, not just budgeted for.

The rollback you do not control: model deprecation

One party can invalidate a policy bundle without touching your release process: the model provider. Providers retire model versions on their own schedule, often with months of notice, sometimes less, and a bundle pinned to a retired model is no longer a rollback target, whatever your runbook says. Treat provider deprecation notices as a standing input to the agentic SDLC: when one lands, every affected bundle is re-run against the successor model on the full held-out adversarial set, gated by the same deploy thresholds as any release, before the cutover date rather than after it. Budget the re-validation as a recurring cost of operating agents rather than an exception, and weigh a provider's deprecation cadence in procurement next to its price per token.

The CTO deliverable: A rehearsed, timed policy-rollback runbook, rehearsed quarterly and at least once before it's needed for a real incident.

Questions for the CTO to consider:

1. If asked tomorrow, could your team produce the exact policy bundle and prompt version that was active when a specific agent action was taken three months ago?

2. Does your release process have a rehearsed rollback path for a policy change, the way it already does for a code deployment?

3. Which stage of your current SDLC would most resist becoming a mandatory gate, Simulation or Safety, rather than an informal check someone can skip under deadline pressure?

Part 7: Leadership patterns for AI-driven change management

Everything covered so far, the roles, the ladder, the teams, the decision rights, the SDLC, is architecture and governance: the parts a CTO can design on a whiteboard. This part is harder, because it's about people who correctly sense that their professional identity is changing, and who are not wrong to feel that way.

The real source of resistance

The resistance a CTO encounters here is rarely about the technology's performance. It's about identity. A significant number of engineers built their sense of professional worth around being the person who writes the code, the one who can sit down and produce the elegant solution. Telling that person their highest-leverage contribution is now reviewing an agent's output rather than writing their own is a loaded statement, whatever the intent behind it. It reads, to them, as a demotion, even when it isn't one.

Leaders who handle this well do not argue the feeling away. They acknowledge it directly, then make the career case concretely: the skill that used to be a nice-to-have, the judgment to know when a solution is subtly wrong, the ability to hold a system's intent in your head while reading someone else's implementation, is now the scarce, well-compensated skill. That is a genuine raising of the ceiling for people who were already good at the parts of the job that were hardest to automate.

What this sounds like in the room

The conversation that actually lands sounds close to this: “The thing you were already best at on this team, catching the bug three layers down that nobody else saw coming, is now the highest-leverage thing you can do, and we're going to pay for it like it is. Your job isn't shrinking. The part of your job that was hardest to automate, and hardest to teach, is becoming the whole job. Here's the title, the level, and the comp band that reflects that, and a date in writing: the band is set aside now, and the title and level land this quarter.” What doesn't work is any version of “your role is evolving” that isn't immediately followed by a specific title, a specific band, and a specific date. And run it only where it is true: Part 2 is clear that some roles do shrink, and a leader who uses this script in a room where headcount is being cut will lose the room for good.

Culture is the control that doesn't show up in an architecture diagram

Andrea Bonime-Blanc, a longtime adviser to boards on governance and technology risk, made two connected points in a 2025 conversation on AI governance: that regulations are the foundation of what you need, but on top of that foundation you need a culture of tech responsibility, and, separately, that how the leader, mostly the CEO but also the board, models or fails to model that culture trickles into the whole organization (Bonime-Blanc, in Global Relay Intelligence & Practice, 2025). Without a culture set by the CEO and reinforced by the board where people are safe to speak up without fear of retaliation, an organization will face serious problems when an agent behaves somewhere it should not.

The practical version for engineering leadership: if an Agent Wrangler or Governance & Safety Engineer spots something concerning in an agent's behavior, they need to be able to escalate it and have the agent's deployment paused, without a career cost for raising the flag. A governance board that exists on paper but punishes the people who use it is worse than no board at all.

Patterns that work

Executives scan for contrast rather than prose. Here is the same guidance as a decision aid:

Instead of this	Do this
Letting the team hear about the AI push through rumor and hallway anxiety	Name the shift yourself, in plain terms, before anyone else frames it for you
Announcing your job is changing and stopping there	Pair every announcement with a visible, funded path into the new roles from Part 3
Rewarding raw output volume the way you always have	Protect and promote the engineers best at judgment, even if they were never the fastest typists
Rolling agents out first on a high-visibility, high-stakes workflow to prove it fast	Prove it small: pick a low-stakes workflow where toil visibly drops without threatening anyone's ownership
Assuming a uniform, gradual team reaction	Expect a bimodal split: some lean in immediately, some disengage. Plan for both, and don't lose the first group by moving too slowly

The five conversations

The script above generalizes one situation. The first quarter of this transition actually runs on five, and they repay preparation the way an incident runbook does. Treat the table as a kit: the situation, what the person in it needs to hear, what you commit to before the conversation ends, and when you follow up.

The situation	What they need to hear	What you commit to	Follow-up
A senior engineer reads the shift as a demotion of the craft their identity is built on	The script above, run sincerely: the judgment they already carry is the scarce skill, and the band will say so	The dated title, level, and band commitment from Part 3	At the band announcement, not after it

The situation	What they need to hear	What you commit to	Follow-up
An early adopter is automating faster than the governance layer can cover	Enthusiasm is not the problem; unscoped enthusiasm is. Their next workflow lands inside a tier, or it waits	A tier-grant path fast enough that compliance never costs more than a week	Weekly, until their fleet is fully registered
A judgment-holder you cannot afford to lose is interviewing elsewhere	The concrete comp path, dated, in writing, this week; a description of future value will not hold them	The Part 3 band, accelerated, plus first pick of the new functions	Within 48 hours, then at each commitment date
A manager's team is shrinking under them	The real headcount answer before their team asks them for it, and what their own role becomes	Redeployment paths named before any reduction is announced	Before any org announcement, then monthly
A staff engineer treats the whole programme as hype	Ownership rather than persuasion: hand them the eval set and ask them to break it	Authority to block a deploy gate on evidence they produce	At the next release gate

The first-90-days communication runbook

Change lands in the order it is announced, so sequence the announcements the way Part 8 sequences containment: each step has a precondition, and skipping the precondition is how rumor outruns you.

When	What is said	To whom	Must already be true
Week 1	The shift, named plainly: what changes, what does not, and when details follow	All of engineering, live	Leadership agrees on the two dates below
Week 2	The funded path: bands, titles, and the promotion route into the new functions	All of engineering, in writing	The Part 3 rubric is approved and budgeted
Weeks 3-6	The first workflow: what the agent now runs, who supervises it, and how to flag a concern	The owning squad, then everyone	The workflow is live in its tier with a named owner

When	What is said	To whom	Must already be true
Week 8	First results, including what went wrong and what the flags caught	All of engineering	The scorecard has real numbers, good or not
Week 12	The expansion decision, and who moves into the new functions first	All of engineering, live	The Part 9 gate check passes, or the reason for holding is stated

Whether any of this is landing shows up on the Part 8 scorecard rather than in this part: the three adoption rows there measure what this part can otherwise only assert.

What culture change actually requires

Architecture and governance are things a CTO can get right through careful design. Culture is something a CTO gets right through repetition, visible consistency between what's said and what's rewarded, and a willingness to sit with the discomfort of a team that is, correctly, unsettled by how much is changing under them. There is no framework that substitutes for that.

The CTO deliverable: A named, funded promotion path into the roles from Part 3, announced before the first engineer has to ask where their job is going.

Questions for the CTO to consider:

1. Has anyone on your leadership team explicitly named, out loud, to the engineering org, what is changing and why, or is the team currently filling that silence with rumor?

2. Which of your senior engineers is best at judgment rather than raw output, and does your current compensation reflect that?

3. What is the smallest, lowest-stakes workflow you could hand to an agent this quarter to start building trust before touching anything closer to the core product?

Part 8: Architecture for the AI-native enterprise (the foundational controls for safe autonomy)

Everything covered so far, the new roles, the RACI, the SDLC, the culture work, rests on an assumption: that someone actually owns enforcing the boundaries you've designed, on a standing basis, with a name attached. Much of what goes wrong in practice is not a missing technical control. It's a missing owner. This part covers who runs it, the small number of hard controls that owner is accountable for, what to do when a control fails anyway, and how those controls map to the frameworks a regulator or auditor will ask about.

Who runs this: the governance board

The architecture below is only as strong as the standing body that governs it. Stand up an AI Governance Board, chaired by the CTO or the Governance & Safety Engineering lead from Part 3, with fixed seats for the CISO, Legal or Compliance, and the CFO's office, plus a rotating operator seat held by whichever business unit has the highest agent-execution volume that quarter. HR does not hold a standing seat, but is a required liaison whenever the agenda touches the career-ladder formalization from Part 3.

Meet monthly, not quarterly. Agentic incidents compound in weeks, and a board that convenes four times a year is reviewing decisions that are already six sprints stale by the time it sees them. Keep its mandate narrow and concrete: approve new autonomy-tier grants, review the risk dashboard defined below, sign off on policy changes for high-privilege agents, and own the escalation path when an Agent Wrangler or Governance & Safety Engineer flags something the RACI in Part 5 didn't anticipate.

This pattern is already standard practice at scale.

This is not a novel proposal. It is how the largest technology companies already operate: Microsoft has run a formal Aether committee (AI, Ethics, and Effects in Engineering and Research) since 2016, chaired by its chief scientific officer, advising leadership on responsible-AI policies, engineering standards, and sensitive uses (Horvitz, n.d.). Google uses lifecycle oversight that evaluates models before and after deployment rather than a one-time approval (Google, 2026). IBM maintains a standing internal AI Ethics Board reviewing major initiatives and setting company-wide policy (IBM, 2024).

The common thread across all three: governance moved from an abstract ethics statement to a concrete, standing, resourced review process years before most of the industry treated it as necessary. One difference is worth naming rather than glossing: those bodies largely advise on the models their companies build, while the board proposed here approves autonomy grants in your own operations, a narrower and more operational mandate, and if anything an easier one to charter.

Report up, not just across

Bring a two-slide agentic risk summary, built from the scorecard below, to the full executive committee every month, and to the board of directors at least twice a year. If your board already gets a cyber-risk

briefing, agent risk belongs on that same agenda, rather than a separate, lower-profile one that's easy to skip when time runs short.

The agentic operations scorecard

References to a risk dashboard only earn their keep if the dashboard has named fields. These fourteen metrics, nine of risk, two of value, and three of adoption and culture, are the ones this paper has referenced throughout; put them on one page. A dashboard that reports only what agents might cost, and never what they produce, eventually loses the room it was built for: the board will ask what it is buying, and that answer belongs on the same page as the risk.

Metric	What it tells the board	Target	Instrumentation source
Agents by autonomy tier	How much of the fleet sits in sandbox, bounded, or high-privilege scope	Skew toward sandbox and bounded	Agent registry, orchestration layer
Shadow-agent count	Unregistered instances found in the last conformance scan	Zero unresolved, across two consecutive scans, with scan coverage reported alongside	Orchestration-layer discovery scan
Short-lived identity coverage	Share of bounded and high-privilege agents on scoped, short-lived credentials	100 percent	Identity layer's credential issuance log
Policy-rollback mean time to execute	How long a rehearsed rollback actually takes end to end	Under one hour	Timed rehearsal drill, logged manually until automated
Replay-trace reconstruction rate	Share of sampled incidents fully reconstructed without interviewing the engineer who built the agent	100 percent	Observability platform, sampled and reviewed manually
Time-to-detect by tier	How long a policy violation or drift event takes to surface	Under 5 min for high-privilege agents	Observability platform alerting timestamps

Metric	What it tells the board	Target	Instrumentation source
Break-glass events	Dual-approved invariant overrides since the last reporting period	Reviewed individually, going beyond a simple count	Runtime enforcement layer's break-glass log
Eval pass rate	Aggregate score and invariant-tagged failures on the most recent deploy	Meets the Part 6 threshold every release	Evaluation platform's deploy-gate output
Cost per agent-hour / per 1,000 governed actions	The unit economics from Part 2, tracked over time rather than estimated once	No universal benchmark; instrument using the formula worked through in Part 2	Finance, combining gateway/observability billing with headcount allocation
Agent-executed share of merged work	What the fleet is actually producing: share of merged PRs or resolved tickets executed by agents, by tier	Trending up within tier limits	Orchestration layer plus repository and ticket analytics
Value per 1,000 governed actions	The other half of the Part 2 unit economics: toil hours returned or throughput gained, priced against the cost row above	Value line exceeds the cost line, and the gap widens	Finance, from workflow-owner baselines and the same billing sources as the cost row
Voluntary agent usage rate	Adoption without mandate: share of squads using agents where none was required	Rising without being pushed	Orchestration layer, correlated with rollout records
Applications into the new functions	Whether the Part 3 ladder is believed: internal applicants per open function seat	Every seat contested	HR requisition data
Escalation flags raised, and the career cost of raising them	The speak-up culture Part 7 requires, measured rather than asserted	Flags steady or rising; career-cost incidents at zero	Governance board minutes plus attrition review

Peer interoperability: who else has to sign off

None of this sits with the CTO alone, and treating it that way is how governance drifts into theater. The CISO owns the security posture of the identity and runtime layers below and should have veto power over any high-privilege autonomy grant. Legal owns the regulatory read on what an agent is allowed to do in a given jurisdiction, especially anything touching personal or financial data, and should sign off before any high-privilege grant, and before a bounded-tier scope is widened toward personal or payment data. HR, as the standing liaison defined above, owns the career-ladder formalization from Part 3 and attends whenever that is on the agenda. Finance owns the unit economics of the whole effort, since every additional control has an operating cost, and that cost has to be defended in the same budget cycle as headcount.

What this looks like at your size

Everything above describes the model at full scale. Most of it compresses; two things do not. The autonomy tiers and the invariants apply identically to a fleet of three agents and a fleet of three hundred, because they are properties of the actions, not of the org chart. And every agent has a named human owner at every size, because accountability does not amortize. What compresses is the apparatus around those two constants.

Around 30 engineers. No board: a standing 30-minute item in the leadership meeting you already run, chaired by you, with one senior engineer as the named second. The seven functions compress to three hats, and no further: a design hat (Agent Architect, Prompt / Policy Engineer, Constraint Designer), an operations hat (Agent Wrangler, AI Reliability Engineer), and a governance hat (Governance & Safety Engineer, with Human-in-the-Loop escalation), the last held by the CTO personally until the fleet outgrows you. The floor is three owners rather than one, because the designer, the operator, and the auditor of an agent should not be the same person: that separation is the control. Run four scorecard metrics: agent inventory, identity coverage, rollback rehearsed, eval pass rate. The roadmap compresses to two phases: prove one workflow, then govern before adding a second.

Around 150 engineers. A real board, minimally seated: CTO, security lead, a finance partner, legal on call quarterly. The three hats become the first dedicated titles, usually in this order: Governance & Safety Engineer first, because the audit trail is the thing you cannot retrofit, then AI Reliability Engineer, then Agent Wrangler as fleet volume justifies it; the design functions tend to stay embedded in senior engineers' existing roles longer. Full scorecard, quarterly to the exec team. The phases run as written, usually faster.

Past a thousand engineers. The model as this paper describes it, with two additions: the operator seat rotates across business units by execution volume, and multi-unit fleets need the board federated, one central board owning invariants and tier definitions, per-unit review owning grants within them. At this size the constraint is not designing the model but keeping three copies of it from diverging.

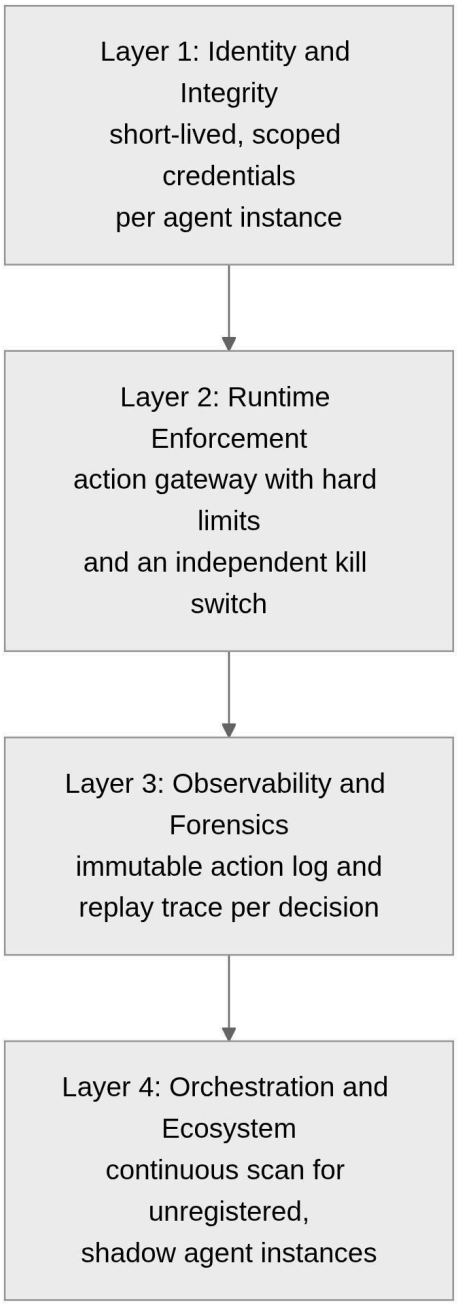
The controls that board is accountable for

The board's job is oversight rather than enforcement mechanics, but it should know what it is actually approving. Four categories of hard, verifiable control sit underneath every autonomy grant. Keep the

mechanism simple. The discipline lies in never skipping one, regardless of how clever any single control is.

Layer	Core mechanism	Threat it mitigates	Owner role
Identity & Integrity	Short-lived, scoped credentials issued per agent instance; no shared service accounts	A compromised or rogue agent acting under another identity's authority	Governance & Safety Engineer
Runtime Enforcement	An AI gateway enforcing hard spending and action limits via policy-as-code, plus a kill switch independent of the agent's cooperation	An agent executing an unauthorized or catastrophic action before a human can intervene	Constraint Designer
Observability & Forensics	An LLM/agent observability platform producing a replay trace for every governed decision	Being unable to reconstruct what happened, or why, after an incident	AI Reliability Engineer
Orchestration & Ecosystem	Continuous inventory scanning for unregistered, shadow, agent instances	The exact fleet-visibility gap behind the 82 percent discovery statistic in Part 1	Agent Wrangler

Applied to the two incidents that opened this paper: PocketOS was an identity-layer failure, an unscoped, long-lived token that the first control exists to eliminate, and Replit was a runtime-enforcement failure, a destructive command no gateway evaluated. Neither required a smarter model to prevent. Each required exactly one control from this table.



[Figure 5. The four-layer control stack, bottom to top.]

Note the distinction from Part 9: the observability baseline in Phase 1 of the roadmap is basic fleet visibility, knowing what agents exist. The replay trace described here, and defined precisely in Part 1, is the mature, Phase 3 version of that same layer, built once you have decided which agents are worth that level of instrumentation.

What to do when a control fails anyway

Replit and PocketOS open this paper for a reason: even a well-designed control layer will eventually fail, through a misconfiguration, a new attack pattern, or a control that was never extended to a particular agent. A CTO needs a rehearsed containment procedure with the same rigor as the prevention architecture above, worked out in advance as a runbook rather than improvised during an incident.

Step	Action	Why	Owner	Depends on
1	Revoke the agent instance's credential at the identity layer, and terminate its in-flight sessions and open transactions, the live-connection gap the Part 1 kill-switch definition names	The fastest containment action available, and one that does not depend on the agent's cooperation	Governance & Safety Engineer	The identity layer from the control table above
2	Freeze the policy bundle version in force	Gives the replay trace a stable reference point; do not let anyone patch the policy mid-investigation	AI Reliability Engineer	Step 1 (credential revoked first, so no further actions execute under the old bundle)
3	Pull the replay trace for the affected window and reconstruct the decision sequence	Acting on the affected system before reconstructing risks destroying the evidence needed to prevent a repeat	Governance & Safety Engineer	The replay trace defined in Part 1

Step	Action	Why	Owner	Depends on
4	Assess blast radius using the tier and scope boundaries from Part 5	An agent correctly confined to a bounded-tier, single-system scope has, by construction, a contained blast radius; this is the payoff of doing the tiering work up front	Constraint Designer	The tiering work from Part 5 already being in place
5	Route the incident through the same review a human-caused incident would get, and add the failure to the held-out adversarial set from Part 6	Closes the loop so the same failure mode is caught in simulation next time	Governance & Safety Engineer	The eval set and its named owner from Part 6

Rehearse this sequence before you need it, the same way you rehearse the policy-rollback runbook from Part 6. An incident response plan that exists only on paper carries the same reliability as the soft guardrails that failed in the first place.

Mapping controls to what a regulator will ask about

The paper promises audit readiness throughout; here is where that promise gets anchored to named frameworks. Treat this as a directional map for a compliance conversation, rather than a legal opinion. The exact clauses that apply depend on your risk classification and jurisdiction; confirm specifics with counsel before representing compliance to a regulator.

Control layer	EU AI Act	NIST AI RMF	ISO/IEC 42001	SOC 2
Identity & Integrity	Traceability and record-keeping expectations for high-risk systems	Govern and Map functions: accountability and role assignment	Resource and infrastructure controls for AI systems	Logical access controls (CC6)

Control layer	EU AI Act	NIST AI RMF	ISO/IEC 42001	SOC 2
Runtime Enforcement	Risk-management and human-oversight obligations for high-risk systems	Manage function: risk treatment	Operational planning and control	System operations (CC7)
Observability & Forensics	Logging obligations for high-risk systems	Measure function: monitoring and metrics	Performance evaluation	Monitoring and incident detection (CC7.2-CC7.3)
Orchestration & Ecosystem	Post-market monitoring obligations	Govern and Manage: third-party and supply-chain risk	Continual improvement	Vendor and risk management (CC9)

Governance as a paved road, not a speed bump

Platform engineering teams learned this lesson a decade before agentic AI arrived (Silver, 2023): a paved road, self-service within well-defined guardrails, moves teams faster than either gatekeeping or a free-for-all. Gartner predicted in 2023 that by 2026, 80 percent of software engineering organizations will have established platform engineering teams (Gartner, 2023).

The same logic applies to agent governance directly. As one 2026 analysis of enterprise AI governance concluded, governance does not slow innovation, it allows AI to grow safely (Goswami, 2026). A hard-coded kill switch means an engineer does not have to personally guess whether an agent's action is safe before shipping it; the boundary already made that decision. That is what a paved road is for: it substitutes a one-time investment in the boundary for a standing tax on every engineer's judgment.

Buildable with tools you already know

Every mechanism in the tables above is buildable with identity, logging, and policy tooling most engineering teams already run in some form, or can buy off the shelf, see Part 6 for the specific tool categories. The barrier was prioritization rather than technical difficulty, and a governance board with a real budget line and a monthly cadence is what keeps it prioritized.

The CTO deliverable: A chartered AI Governance Board with named seats, a monthly cadence, and its first risk dashboard on the calendar.

Questions for the CTO to consider:

1. If your board asked for a two-slide agent risk summary tomorrow, could you produce one, or would that request start a fire drill?

2. Who has veto power over a high-privilege autonomy grant today: the CISO, Legal, both, or no one?

3. Which of the four controls in the table above, identity, runtime enforcement, observability, or orchestration, is your weakest, and who owns closing that gap?

Part 9: The CTO roadmap, a practical journey to AI-native operations

Everything above describes an end state. This final section is about sequencing, because no organization jumps from a few engineers using AI-assisted tools to dozens of governed agents in production in one step, and the ones that try tend to discover Part 8's architecture the hard way, mid-incident.

The timelines below are typical starting points rather than a mandate. Organizations are entering this at wildly different starting points in mid-2026; some are already two phases in, some haven't started. Use it as a sequence, and be candid about which phase you're actually in before committing to the next one. Figure 6 compresses the six phases into the three groupings that matter most for sequencing decisions: a foundation stage spanning Phases 1 through 3, a scale stage that is Phase 4 alone, and a maturity stage spanning Phases 5 and 6.

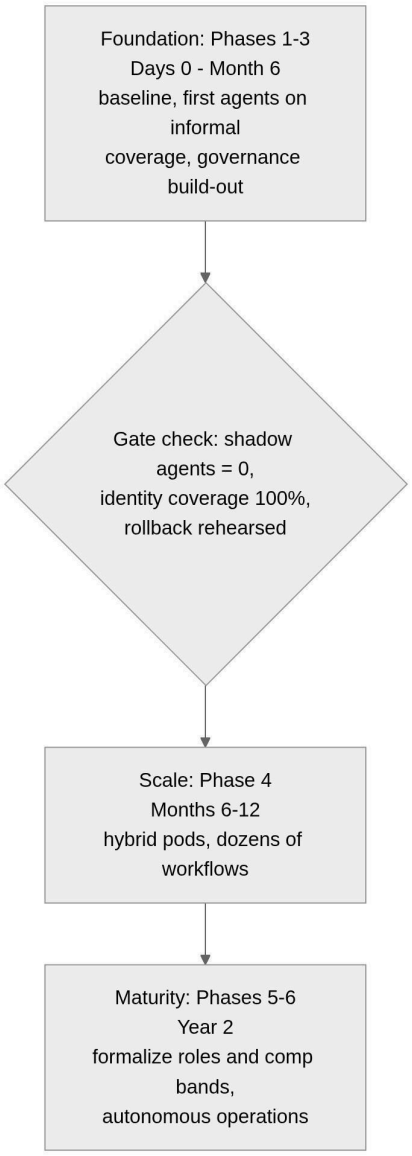
If the apparatus in this roadmap reads as enterprise-scale, Part 8's sizing section shows the compressed forms; the tiers and the named-owner rule travel to every size unchanged.

One sequencing mistake is common enough to call out directly: waiting until Phase 5 to assign the roles from Part 3. You cannot build a governance foundation in Phase 3 or scale to dozens of workflows in Phase 4 without someone actually doing Agent Architect and Governance & Safety Engineer work. Assign that work informally, to existing staff, starting in Phase 2. Phase 5 is not when these roles start existing; it's when you stop treating them as a side responsibility and put a title, a level, and a compensation band behind work people have already been doing for a year. The same applies to the Prompt / Policy Engineer and Constraint Designer duties behind the first gateway and policy-as-code work: the Part 6 tooling table hands those stages owners from Phase 1-2, so interim coverage has to start just as early.

Phase	Typical timeline	What happens
Phase 1: Baseline	Days 0-30	Identify the workflows where agentic execution has the best ratio of leverage to blast radius. Introduce AI-assisted development tools where they aren't already in place. Stand up a basic fleet-visibility baseline: what agents exist and who owns them, not yet the full replay-trace ledger from Part 8.

Phase	Typical timeline	What happens
Phase 2: First supervised agents	Days 30-90	Convert one or two processes, chosen for low blast radius and high volume, to supervised agentic execution. Name interim owners now: a senior engineer covers Agent Wrangler and Human-in-the-Loop Supervisor duties as part of their existing role, informally and without a new title or comp band yet.
Phase 3: Governance foundation	Months 3-6	Build the full control-plane foundation from Part 8, including the replay-trace ledger, and stand up the Governance Board. This requires someone actually doing Agent Architect and Governance & Safety Engineer work day to day; assign it to existing senior staff as an interim responsibility rather than waiting for a formal role to exist.
Phase 4: Scale the squad model	Months 6-12	Extend the squad model from Part 4 across dozens of workflows. The interim role coverage from Phases 2 and 3 is now under real load. Gaps surface fast when governance and architecture are still someone's side responsibility rather than their job.
Phase 5: Formalize the roles	Year 2, H1	Codify what Phases 2 through 4 already forced you to do informally: a leveling rubric, compensation bands, and a real promotion path for the seven roles from Part 3. Hire externally for the roles interim coverage couldn't sustain.

Phase	Typical timeline	What happens
Phase 6: Autonomous operations, human-owned oversight	Year 2, H2 onward	A small number of humans, now formally titled and compensated for it, hold real, well-instrumented oversight over a much larger volume of autonomous execution.



[Figure 6. The roadmap compressed to its three real milestones: foundation, scale, and maturity.]

The gate check before Phase 4

The move that most often goes wrong in this sequence is scaling execution (Phase 4) before the governance foundation (Phase 3) is actually load-bearing rather than merely built. Before authorizing that move, a CTO should be able to demonstrate all of the following as lagging indicators, with real numbers behind each drawn from the Part 8 scorecard rather than intentions or placeholders.

- Zero unresolved shadow agents found in the last two conformance scans.
- 100 percent of bounded and high-privilege agents issuing short-lived cryptographic identities, with no shared service accounts remaining in that population.
- At least one full policy rollback rehearsed end to end, timed, and completed in under an hour.
- Replay trace can reconstruct 100 percent of a sampled set of incidents from the last quarter without needing to interview the engineer who built the agent.
- Time-to-detect for a policy violation or drift event is under an agreed SLA for the tier, for example under five minutes for high-privilege agents, understood as a detection and containment SLA, not a human-approval SLA.

If any of these is missing, the right move is to hold at Phase 3 and fix the gap, instead of scaling around it. The predictable failure mode is scaling execution before the layer that makes that execution accountable is actually in place. The gate check also runs in reverse: if the negative-case signals from Part 2 are present, the move is down a phase, not up one.

Sequencing summary

This is the sequencing summary for a CTO leading AI transformation: governance built ahead of scale, rather than retrofitted after it. Treating Part 8's architecture as a prerequisite for Phase 4, before the incident that would otherwise have made it urgent, is what separates a rehearsed transition from an improvised one.

The CTO deliverable: A signed, dated phase declaration: which phase you are actually in, which gate-check criterion you currently fail, and the date you will reassess, tabled at the Governance Board.

Questions for the CTO to consider:

1. Which phase are you actually in today, not which phase your last board deck claimed?
2. Who on your team is already doing Agent Architect or Governance & Safety Engineer work informally, and when do they get the title and comp band that matches it?
3. If you scaled to Phase 4 tomorrow, which of the five gate-check criteria above would you fail first?

Consolidated deliverables

The nine artifacts this paper asks a CTO to produce, in one place.

Deliverable	Part	Owner	Cadence	Done looks like	Primary tool category
Named owner and documented agent inventory	Part 1	Governance & Safety Engineer (discovery scans run by the Agent Wrangler)	Continuous; reviewed monthly at the Governance Board	Every agent mapped to an owner and an autonomy tier, reviewable in the same meeting as the risk dashboard	Agent registry / orchestration layer
Promotion rubric scoring judgment and review quality	Part 2	CTO / engineering leadership	This quarter, then annual refresh	Rubric scores judgment and review quality with at least equal weight to output volume, in this cycle's promotion packets	n/a (HR process, not a tool category)
Leveling rubric and comp bands for the seven functions	Part 3	CTO, with HR as liaison	Within two quarters of first informal assignment, then annual	All seven roles have a level range and comp-band anchor with no (provisional) tag remaining	n/a (HR / leveling process)
Workflow-to-pattern map with authentication and shared-state design named	Part 4	Agent Architect	Updated each time a new workflow goes live	Every live workflow tagged to one of the three patterns, with authentication method and shared-state design named	Agent identity and attestation (Part 6)

Deliverable	Part	Owner	Cadence	Done looks like	Primary tool category
Published four-category decision-rights matrix	Part 5	Constraint Designer, Governance & Safety Engineer	Same cadence as human access reviews, quarterly recommended	Matrix published with network boundaries and data classifications named, not left as a conceptual diagram	Policy-as-code engine (Part 6)
Rehearsed, timed policy-rollback runbook	Part 6	AI Reliability Engineer	Rehearsed at least quarterly	Rollback rehearsed end to end and timed at under one hour, matching the Part 8 scorecard target	Policy-as-code engine / AI gateway (Part 6)
Named, funded promotion path into the seven roles	Part 7	CTO, with HR as liaison	Announced before first engineer asks	Path named and funded before the first engineer has to ask where their job is going	n/a (HR / leadership process)
Chartered AI Governance Board with first risk dashboard	Part 8	CTO (chair)	Monthly board meeting; summary to exec committee monthly, board of directors twice yearly	Board meets monthly with a two-slide risk summary and its first populated scorecard	LLM / agent observability platform (Part 6)

Deliverable	Part	Owner	Cadence	Done looks like	Primary tool category
Signed, dated phase declaration	Part 9	CTO	Reviewed at every phase transition, and at the board when Phase 4 is proposed	Declaration names the current phase, the failing gate-check criterion, and a reassessment date	n/a (governance process)

Appendix: the CTO readiness assessment

The 28 questions this paper raises, one table, in the order they appear. The per-section clusters remain in place; this is a duplicate, board-ready readiness assessment, not a replacement for reading the sections themselves. Score each question 1 to 5: a 1 means the weak answer in the third column describes you today; a 3 means an owner and a plan exist but no evidence yet; a 5 means you could hand an auditor the artifact this week. An average below 3, or any single 1 on a Part 5 or Part 8 question, is a hold-at-Phase-3 signal under the Part 9 gate check.

Question	Part	What a weak answer reveals	Maturity score (1-5)
If an engineer on your team spun up an agent with production database access tomorrow, would you know about it within the hour, or would you find out the way Replit and PocketOS did?	1	No agent inventory or discovery process exists; shadow agents are your default state, not an edge case.	
Which of your current safety controls are soft guardrails (a system prompt, a policy document) versus hard boundaries (something structurally impossible regardless of what the agent decides)?	1	Your team cannot distinguish enforcement from guidance, so some of your controls may be advisory only.	
Who in your organization currently owns the answer to how many agents you have running in production right now?	1	No one owns it, which means no one is accountable when the count turns out to be wrong.	
If your best engineers were rewarded purely on output volume today, would your top performers already be drifting toward the functions this section describes, or against them?	2	Your incentive structure is actively pushing your best judgment-holders away from the roles you will need most.	
How long does it currently take your team to review a PR you know was AI-generated versus one you know was hand-written, and does anyone track that gap?	2	You are flying blind on the exact bottleneck this paper says will define your velocity.	

Question	Part	What a weak answer reveals	Maturity score (1-5)
Who on your team would you trust today to perform a reasoning audit if a regulator asked you to explain an agent's decision six months from now?	2	No one, which means your first real audit will also be your first attempt at the function.	
What would tell you this programme has stopped paying for itself, and who is watching for it?	2	Nobody owns the cost line, so a programme that has stopped paying will keep running on activity metrics until Finance asks.	
If you mapped your current engineers against the three categories in the table above, how many would already fall into governance and safety without a title that says so?	3	People are doing this work unpaid and unrecognized, and you do not know how many.	
What would you have to change in your compensation bands this quarter to stop your best judgment-holders from feeling like the career ladder ends at senior engineer?	3	Nothing has changed yet, so your best people are still being told the ceiling is where it always was.	
Whose job is it, right now, to decide when a senior engineer is ready to become an Agent Architect?	3	No one's, which means the promotion path this paper asks for does not exist yet.	
For your highest-volume workflow today, could you name which of the three patterns, agent-as-teammate, agent-as-tool-with-memory, or automation mesh, it actually follows?	4	You have not classified your own highest-risk workflow, so you cannot reason about its blast radius.	
If two of your agents disagree about a shared fact right now, would you find out before or after a customer does?	4	Your shared-state design is untested, and the customer is currently your monitoring system.	

Question	Part	What a weak answer reveals	Maturity score (1-5)
Do your agents authenticate to each other the way your services do, or is there an implicit trust relationship you haven't examined?	4	An unexamined trust relationship between agents is an unscoped attack surface.	
Could you produce, this week, a list of every agent in your fleet mapped to sandbox, bounded, or high-privilege, with the specific network boundary and data classification behind each label?	5	Your tiering exists conceptually, not as an auditable artifact.	
Which of your current agent permissions were granted based on a conversation rather than a documented, auditable threshold?	5	Some of your access grants cannot be reconstructed or defended after the fact.	
When was the last time someone reviewed whether a bounded-tier scope from six months ago is still appropriately bounded today?	5	Permissions only ever widen in practice, because no review cadence exists to re-tighten them.	
If asked tomorrow, could your team produce the exact policy bundle and prompt version that was active when a specific agent action was taken three months ago?	6	You cannot reconstruct past agent behavior, which means you cannot defend it either.	
Does your release process have a rehearsed rollback path for a policy change, the way it already does for a code deployment?	6	Your rollback discipline stops exactly where agent-specific risk begins.	
Which stage of your current SDLC would most resist becoming a mandatory gate, Simulation or Safety, rather than an informal check someone can skip under deadline pressure?	6	That stage is your weakest link and the one most likely to be skipped under deadline pressure.	
Has anyone on your leadership team explicitly named, out loud, to the engineering org, what is changing and why, or is the team currently filling that silence with rumor?	7	Rumor is currently doing your change-management work for you, and rumor is always worse than the truth.	

Question	Part	What a weak answer reveals	Maturity score (1-5)
Which of your senior engineers is best at judgment rather than raw output, and does your current compensation reflect that?	7	You can name the person but not the pay gap, meaning your incentives contradict your stated values.	
What is the smallest, lowest-stakes workflow you could hand to an agent this quarter to start building trust before touching anything closer to the core product?	7	If you cannot name one, you are not ready to scale past pilot without a forced, high-stakes first attempt.	
If your board asked for a two-slide agent risk summary tomorrow, could you produce one, or would that request start a fire drill?	8	Your governance reporting is reactive rather than standing, so the board finds out about problems after they happen.	
Who has veto power over a high-privilege autonomy grant today: the CISO, Legal, both, or no one?	8	If the answer is no one, high-privilege grants are currently unilateral.	
Which of the four controls in the table above, identity, runtime enforcement, observability, or orchestration, is your weakest, and who owns closing that gap?	8	If you cannot name an owner, the gap has no closing date either.	
Which phase are you actually in today, not which phase your last board deck claimed?	9	Your internal narrative and your actual maturity have diverged, and the gap is where the next incident hides.	
Who on your team is already doing Agent Architect or Governance & Safety Engineer work informally, and when do they get the title and comp band that matches it?	9	You are relying on goodwill and informal responsibility for your most critical governance work.	

Question	Part	What a weak answer reveals	Maturity score (1-5)
If you scaled to Phase 4 tomorrow, which of the five gate-check criteria above would you fail first?	9	That criterion is your actual constraint on scaling, whatever your roadmap slide says.	

Key takeaways

1. Determinism is gone, and it's not coming back. Agentic systems are probabilistic by nature. The job of engineering leadership shifts from writing correct instructions to architecting the boundaries within which a probabilistic system is allowed to improvise.

2. Review is the new bottleneck, and the new leverage point. When creation is abundant, verification capacity, not headcount, becomes the constraint on velocity, and the career ladder has to reward it accordingly.

3. Decision rights have to be explicit, not assumed. A four-category model, human-only, agent-autonomous, agent-propose-only, and cross-cutting audit, closes the accountability gap that agentic systems otherwise open by default.

4. The unit of deployment is a policy bundle, not a service. Your release process needs a policy rollback path as rigorously tested as your code rollback path.

5. Governance needs a named owner, not just a named framework. A monthly AI Governance Board with fixed seats for the CISO, Legal or Compliance, and the CFO's office, plus a rotating operator seat (HR is a required liaison, not a seat), is what keeps the architecture from decaying into a checklist nobody actually enforces.

6. Sequencing matters more than ambition. Organizations that build the governance layer before scaling execution avoid the incident that forces them to build it afterward.

Related reading

Executive companions to this paper, for readers who want to go deeper on a specific piece.

- The Trustworthy Agentic AI Blueprint (whitepaper.download): a deeper architectural treatment of trustworthy autonomous systems: identity, runtime enforcement, observability, and orchestration, and the operational risk modeling that ties them together.

- The Autonomous Horizon (sakurasky.com): a six-part executive operating-model series: capital allocation, board accountability, and scorecards for the CEO, CFO, COO, CMO, CRO, and CIO.

- The Executive AI Playbook (whitepaper.download): the leadership framework for governing AI and aligning it to business outcomes, for readers who want the cross-functional view beyond engineering.

References

Cited in Harvard style throughout. All sources accessed 1 July 2026 and re-verified 17 September 2026.

Akinyemi, I., 2026. Why AI coding tools shift the real bottleneck to review. LogRocket Blog, 20 January. Available at: blog.logrocket.com/ai-coding-tools-shift-bottleneck-to-review

Cemri, M. et al., 2025. Why do multi-agent LLM systems fail? arXiv (v3, revised 26 October). Available at: arxiv.org/abs/2503.13657

Cursor, 2025. Cursor + Graphite. Cursor Blog, 19 December. Available at: cursor.com/blog/graphite

CodeRabbit, 2025. State of AI vs human code generation report. CodeRabbit Blog, 17 December. Available at: coderabbit.ai/blog/state-of-ai-vs-human-code-generation-report

Cloud Security Alliance, 2026. The shadow AI agent problem in enterprise environments. CSA Blog, 28 April. Available at: cloudsecurityalliance.org/blog/2026/04/28/the-shadow-ai-agent-problem-in-enterprise-environments

Gartner, 2023. Gartner Hype Cycle shows AI practices and platform engineering will reach mainstream adoption in software engineering in two to five years. Gartner Newsroom, 28 November. Available at: gartner.com/en/newsroom/press-releases/2023-11-28-gartner-hype-cycle-shows-ai-practices-and-platform-engineering-will-reach-mainstream-adoption-in-software-engineering-in-two-to-five-years

Elemuwa, F., 2026. How to design multi-agent memory systems for production. Mem0 Blog, 3 March. Available at: mem0.ai/blog/multi-agent-memory-systems

Faros Research, 2025. The AI Productivity Paradox Report 2025. Faros AI Blog, 23 July. Available at: faros.ai/blog/ai-software-engineering

Gartner, 2025. Gartner predicts 40% of enterprise applications will feature task-specific AI agents by 2026, up from less than 5% in 2025. Gartner Newsroom, 26 August. Available at: gartner.com/en/newsroom/press-releases/2025-08-26-gartner-predicts-40-percent-of-enterprise-apps-will-feature-task-specific-ai-agents-by-2026-up-from-less-than-5-percent-in-2025

Global Relay Intelligence & Practice, 2025. Transcript: Andrea Bonime-Blanc and Ryan McManus podcast. GRIP, 26 November. Available at: grip.globalrelay.com/transcript-andrea-bomine-blanc-and-ryan-mcmanus-podcast

Google, 2026. AI principles: AI governance across the model lifecycle. Google AI, accessed 1 July 2026. Available at: ai.google/principles

Goswami, R., 2026. From principles to practice: what AI governance actually looks like in 2026. CTO Magazine. Accessed 17 September 2026. Available at: ctomagazine.com/ai-governance-for-ctos-2026-beyond

Horvitz, E., n.d. Aether Committee at Microsoft. Accessed 17 September 2026. Available at: erichorvitz.com/Aether_Committee_Microsoft.htm

Hughes, C., 2026. System prompts are not security controls: a deleted production database proves it. Zenity Blog, 28 April. Available at: zenity.io/blog/current-events/ai-agent-database-deletion-pocketos

IBM, 2024. A look into IBM's AI ethics governance framework. IBM Think, accessed 1 July 2026. Available at: ibm.com/think/insights/a-look-into-ibms-ai-ethics-governance-framework

Nolan, B., 2025. AI-powered coding tool wiped out a software company's database in 'catastrophic failure'. Fortune, 23 July. Available at: fortune.com/2025/07/23/ai-coding-tool-replit-wiped-database-called-it-a-catastrophic-failure

Qodo, 2025. State of AI code quality 2025. Qodo Report, June. Available at: qodo.ai/reports/state-of-ai-code-quality (survey of 609 developers)

Silver, A., 2023. Building paved paths: the journey to platform engineering. Engineering@Microsoft, 15 November. Available at: devblogs.microsoft.com/engineering-at-microsoft/building-paved-paths-the-journey-to-platform-engineering

TechCrunch, 2025. Cursor continues acquisition spree with Graphite deal. TechCrunch, 19 December. Available at: techcrunch.com/2025/12/19/cursor-continues-acquisition-spree-with-graphite-deal