STRATEGIC WHITEPAPER

The Enterprise Agentic Software Factory

Why AI Coding Stalls Without an Operating-Model Redesign, and the Governed Model That Turns Agents into Measured Delivery

Yassir Kachri | Enterprise AI, Cloud & Security Architect | 2026

CONTENTS

	Executive thesis	2
1	Diagnose: the delivery bottleneck and the three debts	4
2	Design: five planes and two human gates	7
3	Transform: roles, skills and brownfield adoption	11
4	Govern: reversibility and unit-cost FinOps	15
5	Execute: ninety days, Day-2 resilience and the executive checklist	18

EVIDENCE STANDARD

This paper reports no client results. Every number in it is an external figure with its source cited, a planning assumption stated as such with its arithmetic, or a policy threshold labeled as a recommended starting value. The mechanisms, configurations and sources behind each section are in the companion reference guide, *The Enterprise Agentic Software Factory*. Web sources were consulted in fall 2026.

ABOUT THE AUTHOR

Yassir Kachri is an Enterprise AI, Cloud & Security Architect with more than 25 years of experience designing, securing, and operating mission-critical systems for large regulated organizations across AWS, Microsoft Azure, and Oracle Cloud. He is the creator of **Kosty** (an open-source AWS security and FinOps auditing CLI) and co-creator of **ConfigLens**.
yassirkachri.com · linkedin.com/in/yassirkachri · github.com/yassirkachri/enterprise-agentic-software-factory

© 2026 Yassir Kachri. All rights reserved.

EXECUTIVE THESIS

AI Coding Stalls Unless the Delivery Operating Model Changes

> Most enterprises bought AI coding as a productivity tool for individual developers and measure it at the keyboard. The return shows up somewhere else. A change creates value when it reaches production and stays there, and the stages that decide whether it does so wait on people. Agents that write code faster send more work into exactly those stages, so the rollout is an operating-model decision with a tooling component attached.

Faster Coding Raises the Delivery Ceiling Only a Little

Amdahl's Law, first stated for parallel computing, bounds the speedup of any system in which only part of the work gets faster.[1] Applied to software delivery, the accelerated part is coding. Clarifying what a change should do, reviewing it, waiting for a release, and repairing a failed deployment all wait on a person. If coding is a share p of the time from a ready ticket to production and becomes s times faster, the whole path speeds up by at most $1 / ((1 - p) + p / s)$. The arithmetic holds for any organization, so the table uses round values of p rather than anyone's measurements:

Coding share of delivery time	Coding twice as fast	Coding infinitely fast
20%	1.11x	1.25x
30%	1.18x	1.43x
40%	1.25x	1.67x
50%	1.33x	2.00x

These ceilings assume the other stages stay as they were, and the published evidence suggests they grow instead. Faros AI's 2025 study of more than 10,000 developers found that teams with high AI adoption merged 98% more pull requests while review time per pull request rose 91%, and observed "no significant correlation between AI adoption and improvements at the company level."[2] Faros sells engineering analytics and the study has no control group, so I read it as directional. Its direction matches

[1] Gene M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *AFIPS Spring Joint Computer Conference*, 1967.

[2] Faros AI, "The AI Productivity Paradox Report 2025," July 23, 2025, https://www.faros.ai/blog/ai-software-engineering.

DORA's 2025 finding that AI acts as "an amplifier, magnifying an organization's existing strengths and weaknesses."[3] An organization whose review stage was already its constraint should expect agents to amplify that constraint.

+98% / +91%
Merged pull requests and review time per pull request, teams with high AI adoption (Faros AI, 2025)

1.43x
Delivery speedup ceiling when coding is 30% of lead time and becomes infinitely fast (Amdahl's Law)

Amplifier
DORA's 2025 description of AI's role: it magnifies existing strengths and weaknesses

A Governed Factory Moves Judgment to Two Gates and Measures Each Service Against Itself

The design that follows runs at Level 3 conditional autonomy. Agents carry a work item from an approved specification to a pull request, and a named Change Owner per domain decides twice: approving the intended change before work starts and accepting the evidence before it merges. Five planes do the work between those gates. Reviewers become Change Owners who judge evidence, services join in waves that open only when their people can run the gates, decision rights follow how hard a choice is to undo, and spend is governed by what a merged change costs.

BOTTOM LINE

Delivery speed is capped by the stages that wait on people. Agents accelerate coding and add load to intent, review, and rework, which already set the pace.

The return comes from an operating model. Five governed planes and two human gates turn agent output into changes a named person can stand behind.

Each service is measured against its own baseline. A quarter of delivery metrics, rebuilt from records the organization already keeps, comes before funding; lead time and rework then decide whether a service keeps its autonomy, and agent cost per merged change shows what the factory spends to get there.

[3] DORA, "State of AI-assisted Software Development 2025," https://dora.dev/research/2025/dora-report/.

Uncoordinated AI Moves the Bottleneck to Intent, Review and Rework

> Agents speed up the stage where they work and push larger, busier changes into the stages around it. Ungoverned, that output runs up technical, intent, and cognitive debt faster than a team can pay it down. Level 3 conditional autonomy is the operating point that keeps a named person accountable for every production change while machines do the work and the checking in between.

Output Rises Where the Agent Works, and Review Absorbs It

No single study settles how agents change enterprise delivery, but four bodies of evidence, weighed by what each measured, agree on a mechanism. Telemetry shows output rising and review absorbing it. In the Faros AI data, high-adoption teams also saw average pull request size grow 154%.[4] Each percentage compares a team's highest-adoption quarters with its own lowest, which leaves no control group, and a 2026 follow-up reporting time in QA up 300.6% does not state its comparison base.[5] Both reports give the direction of the effect; neither gives a planning number.

Review findings per change rise with AI co-authorship. CodeRabbit compared 470 open-source pull requests and found that AI co-authored changes averaged 10.83 issues each against 6.45 for human-only ones, about 1.7 times as many.[6] Authorship was inferred, the vendor sells AI code review, and counts per pull request cannot separate size from quality, but the extra findings land on a reviewer who now receives larger changes.

Felt speed and measured speed diverge. In METR's 2025 randomized trial, 16 experienced developers took 19% longer with AI tools allowed, yet believed afterward that the tools had made them 20% faster.[7] METR has since marked those results out of date and calls its newer data "an unreliable signal,"[8] so the

[4] Faros AI, "The AI Productivity Paradox Report 2025," July 23, 2025, https://www.faros.ai/blog/ai-software-engineering.
[5] Faros AI, "The Speed Trap: 8 takeaways from our latest AI engineering research," September 18, 2026, https://www.faros.ai/blog/ai-speed-trap-takeaways.
[6] CodeRabbit, "State of AI vs Human Code Generation Report," December 17, 2025, https://www.coderabbit.ai/blog/state-of-ai-vs-human-code-generation-report.
[7] METR, "Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity," July 10, 2025, https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/.
[8] METR, "We are Changing our Developer Productivity Experiment Design," February 24, 2026, https://metr.org/blog/2026-02-24-uplift-update/.

study says nothing about the effect today. It does show how far people misjudge their own speed, an error inherited by any enterprise that funds agents on self-reported time savings.

DORA measures the system, and deployment rework rate is the metric agents test hardest: "the ratio of deployments that are unplanned but happen as a result of an incident in production."[9] A quarter of each service's path from ready ticket to production, split at events every Git host already records and taken before agents expand (my recommended starting window), shows which waiting stage is largest and forms the baseline for every later claim.

Ungoverned Agents Run Up a Triple Debt at Machine Speed

Ungoverned agent output creates debt faster than a team pays it down, and each kind has its own signal and control:

Debt	What accumulates	Early signal	Control that retires it
Technical	Code that works but is hard to change	Rework rate; churn	Deterministic checks; four-station verification
Intent	Behavior nobody wrote down or approved	Clarification loops; changes with no specification	Durable specifications and a Spec Gate
Cognitive	Code no current engineer understands	Review time per line; fast approval of large changes	A Merge Gate on evidence; a critic that did not write the change; reviewer skills

A rounding rule that lives in one engineer's memory is intent debt: no agent can follow it and no reviewer can check it, so it surfaces in production as a rework deployment. A large agent-written change approved quickly because the queue behind it is long is cognitive debt: the organization now owns code it does not comprehend. Agents accrue both faster than a human team, because they produce more change per hour and never hesitate over a rule nobody told them.

[9] DORA, "DORA's software delivery performance metrics," last updated January 5, 2026, https://dora.dev/guides/dora-metrics/.

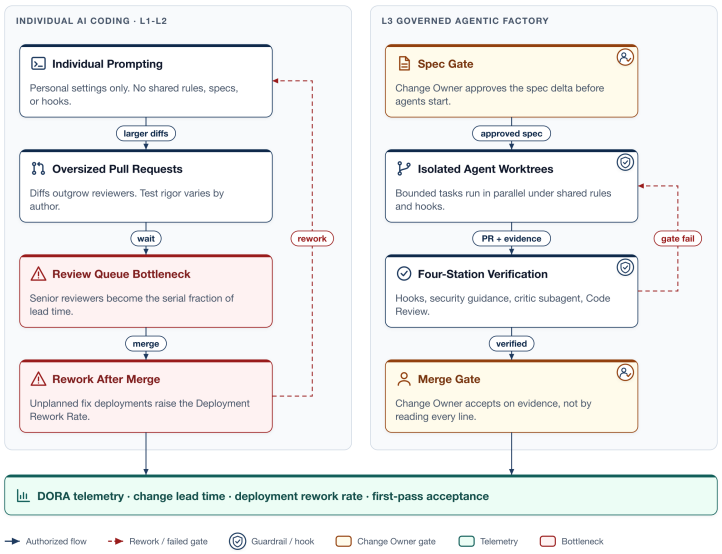

Exhibit 1. Individual AI coding sends oversized changes into a human review queue and loops rework back after merge, while a Level 3 factory moves human judgment to a Spec Gate and a Merge Gate and runs machine verification between them.

Level 3 Conditional Autonomy Is the Standard for Regulated Production

A five-level scale, a framework for this paper and no product setting, states who decides what:

Level	Agent scope	Human role
L1 Assisted	Suggestions and chat	Author of every change
L2 Supervised	Edits in one session, one task	Approves each step
L3 Conditional	**Ticket to pull request inside shared rules and gates**	**Change Owner approves the specification and the merge**
L4 High	Merges low-risk change classes	Audits samples after the fact
L5 Full	Plans and ships without approval	Sets objectives only

Most organizations that adopted agents one developer at a time sit at L1 and L2 without shared rules. I recommend L3 as the enterprise default and the ceiling for regulated production, because each production change needs a named approver and an agent cannot hold that accountability; an agent generates new steps for every change, so it cannot earn a standard change's pre-authorization. Organizations without that obligation can run low-risk classes at L4 in the same factory.

> **LEVEL 3 HOLDS ONLY WHILE ITS CONDITIONS HOLD**
>
> **An agent that cannot converge goes back to a person** at L2 once it exhausts its attempt budget.
> **Ambiguous intent returns to the Spec Gate,** where the Change Owner decides the rule; no agent guesses at it.
> **Rising rework lowers autonomy:** rework above baseline for two consecutive weeks (my recommended starting window) drops the service to L2 until a corrective change is approved.

Five Planes and Two Human Gates Turn Agents into a Governed Factory

> The factory is organized as five planes, each with one job and a named operator, and each handing the next an artifact it can check. People make two decisions per work item, and machines do the checking in between, so a reviewer's attention goes only where a person adds judgment.

Each Plane Hands the Next an Artifact It Can Check

Plane	Capability	Risk it retires	Operated by
1 Intent	Ticket changes become approved changes to durable specifications	Business rules lost in ticket threads	Change Owner
2 Context and workbench	Each agent works under central policy, scoped rules and an isolated workspace	Agents weakening their own guardrails	Platform team
3 Verification	Four stations check the output before the Merge Gate	Self-certification	Change Owner, AppSec
4 Headless CI/CD	Agents run in pipelines with short-lived identity in a sandbox	Credentialed agents steered to leak secrets	Platform team, AppSec
5 Gateway and telemetry	Every model call is routed, metered and joined to delivery data	Unattributed spend; unmeasured outcomes	Platform team, budget owner

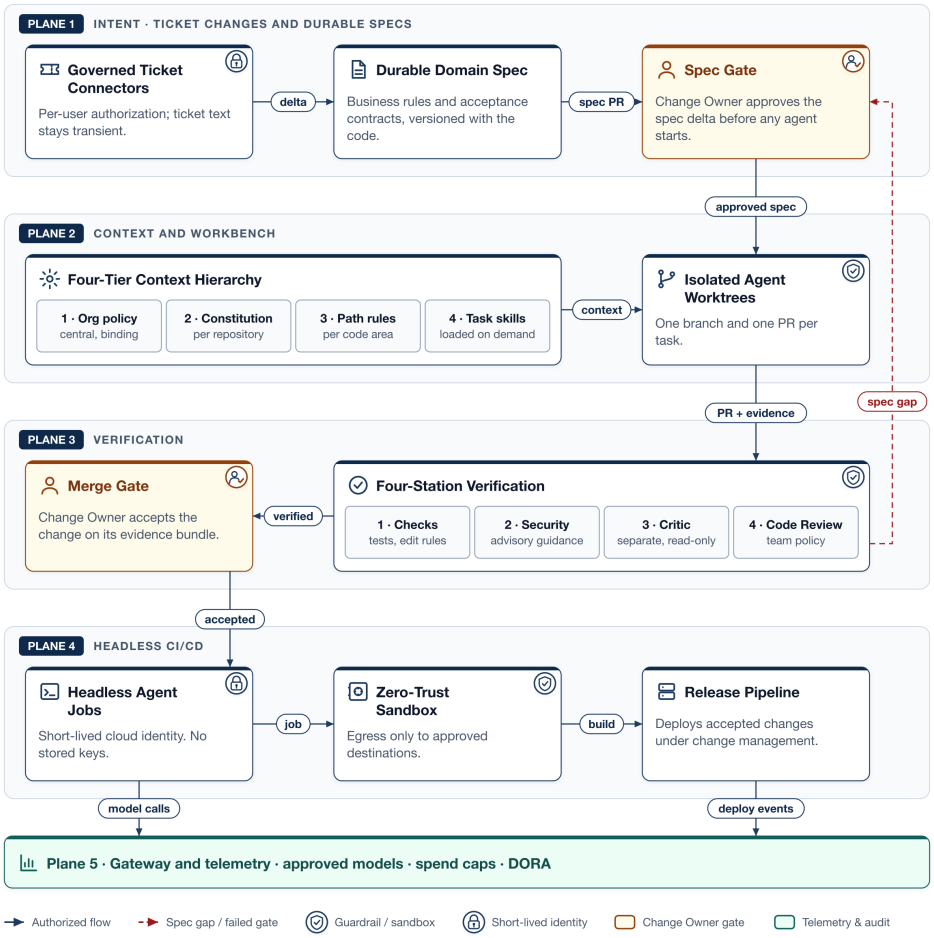

Exhibit 2. Each plane hands the next an artifact it can check, an approved specification, a branch built inside guardrails, an evidence bundle, and a sandboxed, attributable job, while one gateway and telemetry plane observes all four.

The Change Owner Decides Twice, Before Work Starts and Before It Merges

Spec Gate: before any agent starts

- The Change Owner approves the change to the domain's specification that the ticket implies.
- Ambiguity is settled here by a person, before code exists.
- A Spec Gate wait of one business day at the 90th percentile is my recommended starting target.

Merge Gate: before the change merges

- The Change Owner accepts the change on its evidence bundle: the approved specification change, required test results, the critic's verdict, and every review finding resolved.
- The full diff is read when a stated trigger applies, such as a path that moves money.

The Change Owner is an engineer accountable for one domain, such as a payments API, who neither writes most of the code nor reads every line of it. A change flagged as high risk also waits for a security champion's deeper scan, the one other human wait the design accepts.

Intent Lives in Durable Specifications, Never Only in Ticket Threads

A rule that exists only in a ticket comment cannot be checked by anyone. Plane 1 turns each ticket into a change to a durable domain specification, versioned with the code it governs, with business rules written as cases and expected outcomes. Agents read tickets through governed connectors that authenticate each user with OAuth 2.1, per the MCP authorization specification.[10] Legacy services gain specifications where they change.

Central Policy and Deterministic Checks Keep Agents Inside Their Guardrails

An agent told to make the tests pass can edit the tests. Plane 2 gives each agent organization policy that no team can override, rules per repository and code area, and skills loaded on demand, with each task in its own isolated workspace. Deterministic checks refuse writes to protected paths such as contract tests and fail closed, so a check that crashes blocks the edit. A test gate holds the agent until the suite passes, within an attempt budget after which the item goes back to a person.

Verification Counts Only When It Is Independent of the Author

The session that wrote a change never declares it done, and the four stations differ in how independent of the author they are:

Station	Why the author cannot sway it	Can it block the merge?
1 Deterministic checks and tests	Rules run as code, outside the model	Yes
2 Security guidance	Flags risky patterns while the agent writes	No; advisory
3 Independent critic	A separate, read-only session judges the diff against the approved specification, with rules read from the protected main branch	Yes, as a required check
4 Policy-based code review	Review rules come from the base branch, which the change cannot alter	Yes, through branch rules

The stations apply to every pull request on a Level 3 service, whoever or whatever wrote it, so the choice of tool never becomes a way around the gates.

[10] Model Context Protocol Specification, "Authorization," https://modelcontextprotocol.io/specification/latest/basic/authorization.

Pipelines Run Agents Without Standing Credentials

An agent that reads untrusted content while holding standing keys can be steered into leaking them. Plane 4 gives each agent job short-lived federated identity, stores no provider key anywhere, withholds push rights from any job that runs a model, and confines execution to a sandbox that reaches only approved destinations.

THE PIPELINE RISK HAS A PRECEDENT

On August 26, 2025, malicious versions of several Nx packages were published to npm. They ran a post-install script that scanned systems for sensitive data and "attempted to use local AI tools (like Claude and Gemini)."[11] The same script runs wherever a package install runs, including a CI runner, so an agent on any machine that holds credentials is a capability an attacker can borrow.

One Gateway Attributes Every Model Call to a Person, a Repository and an Outcome

Engineers sign in through the corporate identity provider, the gateway applies an approved model list and spend caps per person and group, and one telemetry collector joins spend to each service's delivery metrics. One documented failure mode deserves executive attention: unless configured to merge, settings delivered by the gateway silently replace the central deny rules already on laptops.[12] I gate every fleet rollout on proof that a central deny rule still refuses.

[11] Nx, s1ngularity post-mortem, September 5, 2025, https://nx.dev/blog/s1ngularity-postmortem.
[12] Claude Code documentation, "Deploy managed settings," https://code.claude.com/docs/en/managed-settings.

Reviewers Become Change Owners, and Adoption Moves in Waves Gated on People

> The controls define the new work without teaching it. Where roles and habits stay as they were, agents produce changes faster than people trained to read every diff can accept them, and the review queue grows. Autonomy is therefore raised per service, and only once that service's people can run the gates it depends on.

The Reviewer's Job Shifts from Reading Diffs to Judging Evidence

Role	Work at Level 3	New skill
Change Owner	Approves specification changes; accepts or returns changes on evidence	Specification writing; reading the evidence bundle and the scorecard
Domain engineer	Turns tickets into specification changes; drives agents; takes items back at L2	Workbench habits; specification writing
Product owner or analyst	Writes the ticket change the specification starts from	Writing rules as cases with expected outcomes
Security champion	Runs the deep scan on high-risk changes; answers each finding in writing	Judging findings from a nondeterministic scan
Platform engineer	Runs the factory as an internal product	Operating a tier-1 internal service
Budget owner	Sets caps and overrides from attributed spend	Reading cost per merged change

None of these is a new job title. The Change Owner is usually the tech lead who already owns a domain informally; what changes is the written expectation and the time the role is given, and a tech lead named Change Owner without relief from other work becomes the queue the evidence predicts. Reading an evidence bundle is taught as an order of attention: the approved specification change, the critic's verdict, any weakened or deleted test, unresolved review threads, and the full diff only when a stated trigger applies, such as a path that moves money or a migration.

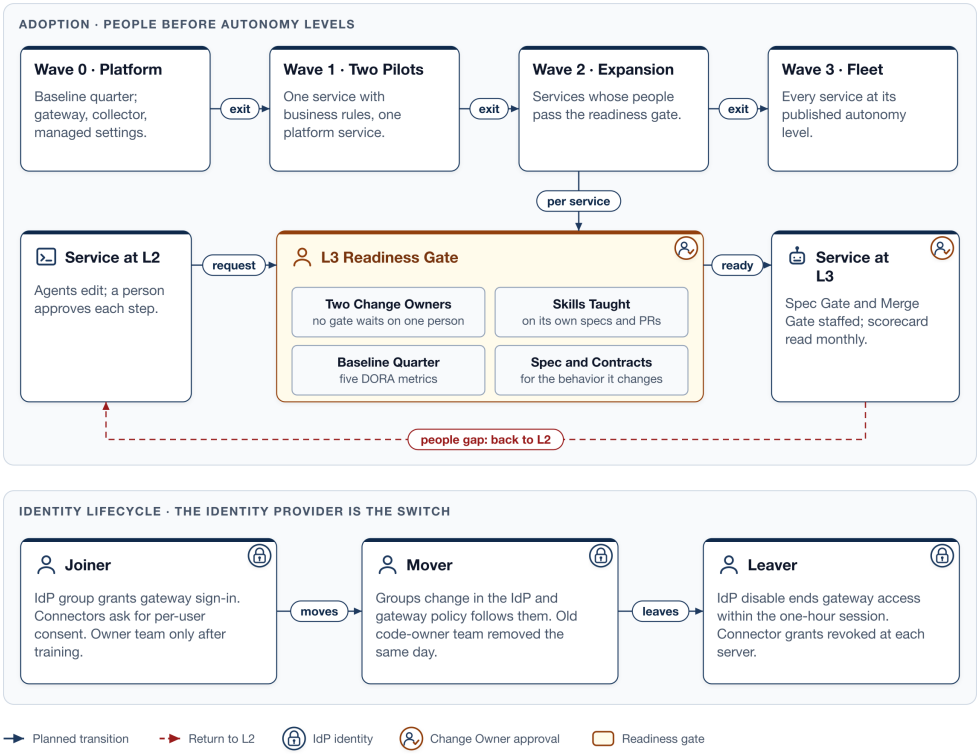

Exhibit 3. A service moves from L2 to L3 only through a readiness gate on its people, adoption proceeds in waves that end on per-service exit criteria, and identity follows each person from joining to leaving.

Two Trained Owners per Domain Keep Every Gate Staffed

Both gates wait on a person by design, so a domain with one Change Owner stops whenever that person is ill, on leave, or in a long meeting. My recommended starting rule is that no service reaches L3 with fewer than two trained people holding the Change Owner right. The floor covers absence and says nothing about load; Spec Gate wait and review wait show when a domain needs a third owner or a split. Both rules slow the first waves, and I accept that cost because autonomy granted before anyone can judge its output produces cognitive debt, paid later when a change nobody understands fails in production.

Services Join in Waves, and Each Wave Ends on Measured Exit Criteria

Wave	Scope	The wave ends when
0 Platform (days 1-30)	Gateway, telemetry, central policy and checks; baselines rebuilt for the pilots	Every pilot engineer signs in through the gateway; central policy is proven to survive it; baselines are published
1 Pilots (days 31-60)	Two services at L3	Each pilot meets its readiness exit criteria
2 Expansion (from day 61)	Services added a few at a time, paced by training capacity	Each added service meets its exit criteria; any that fail return to L2
3 Fleet	Every remaining service assessed	Every service has a published autonomy level and a downgrade path

A service's readiness record names at least two trained Change Owners and a security champion, and confirms that its baseline, specifications and branch rules exist. It exits its wave when lead time and rework are at or better than its own baseline, review wait is at or below baseline, first-pass acceptance has been reported for four weeks running, and Spec Gate wait is within one business day at the 90th percentile; the four-week window and the one-day target are my recommended starting values. Review wait is on the list because a pilot whose lead time holds while its pull requests wait longer has shifted the constraint onto its Change Owners. One pilot has real business rules and the other is an internal platform service, so the platform team feels the factory's friction first; the most critical service waits for a later wave.

Brownfield Services Coexist Under One Set of Branch Rules

Most teams will run at L1 or L2 for a long time, and coexistence needs no second rulebook because autonomy is set per service, never per person. The same engineer, laptop and sign-in serve an L3 settlement service and an L2 legacy batch system; the repository's specifications and branch rules make the difference. Legacy services are blocked first by missing specifications, then by thin tests, so they stay at L2 while supervised agents write characterization contracts that pin current behavior, each approved at the Spec Gate. That is where an L2 agent pays for itself on legacy code. Wave 3 means every service has been assessed, which differs from every service running at L3.

Identity Follows Each Person from Joining to Leaving

Every right in the factory derives from the corporate identity, and gateway access expires within the session lifetime, one hour by default.[13] Connector tokens are not revoked at each server when a person is removed, and a mover is the case organizations handle worst, because nothing breaks when it is missed:

Event	What changes	Control
Joiner	Gateway sign-in, model list and caps by group; ticket connectors	Identity-provider groups; per-user consent per connector
Joiner as Change Owner	Approval right at both gates	Owners-team membership, granted only after training
Mover	Policies, caps and approval rights	Group change in the identity provider; removal from the old owners team the same day, checked monthly
Leaver	Gateway access; connector grants	Disabling the identity ends gateway access within the session lifetime, one hour by default; grants revoked at each authorization server
Leaver	Agent jobs	Nothing to change: pipeline jobs use repository identity, never a person's

[13] Claude Code documentation, "Claude apps gateway for Amazon Bedrock, Claude Platform on AWS, Google Cloud, and Microsoft Foundry," https://code.claude.com/docs/en/claude-apps-gateway.

Decide by Reversibility, and Govern Spend by Cost per Merged Change

> Each decision belongs to one accountable role, at the lowest level that bears its consequence, with a recorded way back. Choices a commit can undo are delegated; choices that cannot be undone go to the Architecture Review Board. Every exception ends on a date, and spend is judged by the agent cost of each change that reaches production.

Decision Rights Climb with the Cost of Undoing Them

Band	Examples	Undone by	Decided by
Two-way	A spend cap, model routing for a role, the review depth of one repository	A reviewed commit	One delegated owner: budget owner, platform team, Change Owner
Data flow or autonomy	Raising a service to L3, a new egress domain, a new ticket connector, transcript retention	Closing the flow or lowering the level; what crossed stays crossed	The role answering for the blast radius: Change Owner or AppSec lead
One-way	Enabling managed code review for the organization; provider, region, cross-provider failover	A contract change, on the provider's schedule	Architecture Review Board

Routing every decision through the board would turn it into the serial stage the diagnosis measured, so the board meets every two weeks (my recommended starting cadence), never sees reversible decisions, and records its outcomes as commits. The safe direction is open to more roles than the risky one: raising a service to L3 needs its Change Owner and a readiness record that holds, while the Change Owner, the platform on-call or the AppSec lead may each lower it to L2 alone. No role may raise a regulated production service to L4 by any route.

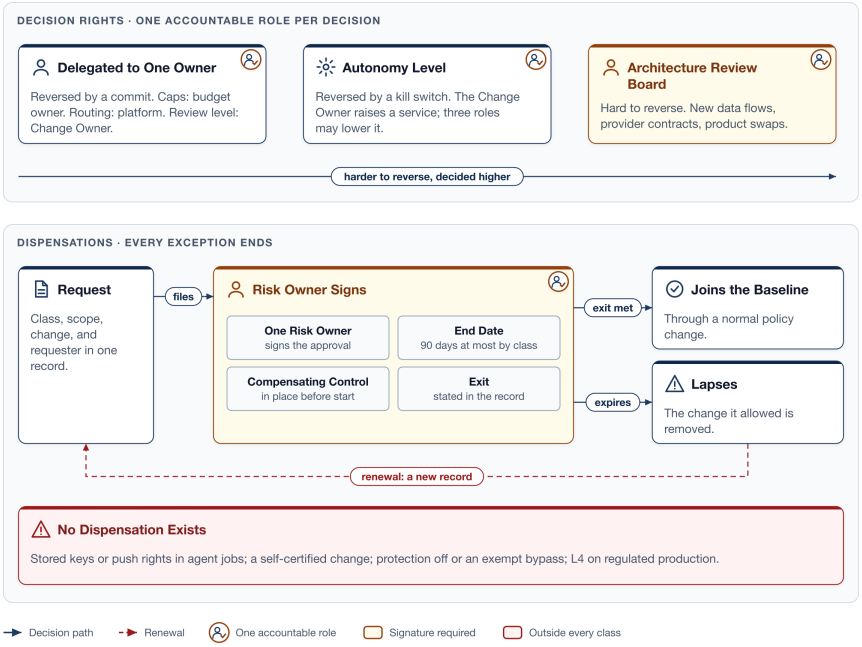

Exhibit 4. Decisions go to the lowest role that bears their consequence and climb toward the board as they become harder to reverse, while every dispensation carries one risk owner, an end date, a compensating control and an exit.

Every Decision Has Exactly One Accountable Role

Decision	Accountable	Responsible or consulted
Spec Gate and Merge Gate	Change Owner	Product and AppSec consulted
Override a critic failure	Change Owner, never the approving one	Platform consulted
Raise a service to L3	Change Owner	Platform responsible; AppSec consulted
Central policy, checks and sandbox	AppSec	Platform responsible; Change Owner consulted
Caps, group caps, overrides	Budget owner	Platform responsible
Provider, region, failover; dispensation classes	Architecture Review Board	All roles consulted

Every Exception Carries an Owner, an End Date and an Exit

A request that does not fit the baseline, such as a new egress domain or a connector the fleet does not yet include, becomes a dispensation, TOGAF's term for a sanctioned deviation.[14] Each class the board defines names one risk signer, a longest term, a compensating control in place before the start, and an

[14]The Open Group, *The TOGAF Standard, 10th Edition*, 2022: building blocks, architecture governance, the Architecture Board, and dispensations. No text is quoted.

exit. My recommended starting terms are 90 days for egress, connector and isolation exceptions and 30 days for a service running at L3 before one readiness check holds, with one renewal. A daily check fails each record the day after it expires. Controls are also kept from being written for one vendor: each is a TOGAF building block with a swap test any product must pass, such as “a branch without the owner’s approval on the specification cannot merge.”

FOUR CONTROLS ADMIT NO DISPENSATION

- A stored model-provider key, or a push credential, in a job whose agent runs a model.
- A change certified only by the session that wrote it.
- Branch protection or the critic’s rules turned off on a Level 3 repository, or a bypass that leaves no record.
- Level 4 or 5 on a regulated production service.

Each produces something that cannot be called back: a leaked key has been used, an unreviewed change is in production, and an unrecorded bypass cannot be counted.

Spend Is Governed by Agent Cost per Merged Change

Agent cost per merged pull request is laptop, CI and managed code review agent spend divided by merged pull requests, per service and quarter. The executive committee’s quarterly question is whether lead time and rework moved against the service’s own baseline, and at what agent cost per merged change. The figure sits beside first-pass acceptance, and neither is a target on its own, since targets raise “the likelihood that teams will try to game the metrics.”[15] Unit cost also exposes a trap a token price hides: a cheaper model that fails its first pass more often re-runs the agent, the critic and the pipeline, so cost per merged change can rise while the token bill falls.

The Claude Code cost documentation states that “across enterprise deployments, the average cost is around $13 per developer per active day and $150-250 per developer per month.”[16] Taken as a planning assumption, that range gives a first envelope of engineers × $150-250 per month:

$15,000-25,000
per month for 100 engineers (100 × $150-250)

$37,500-62,500
per month for 250 engineers (250 × $150-250)

$150,000-250,000
per month for 1,000 engineers (1,000 × $150-250)

The source states no sample or period, so I read it as developers’ own sessions and budget CI jobs, managed code review and the gateway separately. Telemetry must reconcile with each provider bill within 10% (my recommended starting tolerance) before anyone trusts the unit cost, and lines generated or self-reported time saved stay off the scorecard. Finance values the return with its own cost of a day of lead time and of a rework deployment.

[15] DORA, “DORA’s software delivery performance metrics,” last updated January 5, 2026, https://dora.dev/guides/dora-metrics/.
[16] Claude Code documentation, “Manage costs effectively,” https://code.claude.com/docs/en/costs.

Ninety Days, Three Exit Gates, and a Kill Switch for Every Scope

> The first ninety days run in three windows, and each window ends at an exit gate measured against the pilots' own baselines. From the first pilot onward, the factory itself is operated as a tier-1 service: its changes ship through rings, and autonomy drops through graduated kill switches that named roles have drilled.

Each Thirty-Day Window Ends at an Exit Gate

Days 1-30	Days 31-60	Days 61-90
Rebuild the baseline quarter for two pilot services; deploy the gateway and telemetry; ship central policy and checks.	Level 3 on the pilots with downgrade triggers live: Spec Gate, four stations, pipeline jobs, spend caps; a canary ring for factory changes.	Add two services; first quarterly readout to the executive committee.
EXIT GATE Every pilot engineer signs in through the gateway; central policy survives the gateway; baselines published.	**EXIT GATE** Each pilot meets its readiness exit criteria; no unattributed spend; telemetry within 10% of the invoice.	**EXIT GATE** Lead time and rework at or better than baseline per service; autonomy confirmed or lowered.

The baseline quarter, the two pilots, the two added services and the 10% tolerance are my recommended starting values. A pilot that misses its exit criteria stays in its window until it meets them.

Factory Changes Reach the Fleet Through Rings, Promoted on Evidence

A new agent release, model, check or critic instruction changes how every team's agents behave without touching application code, so each ships through rings: the platform team's own repositories for one week, a canary group from the pilot services for two weeks (both my recommended starting values), then the fleet. Promotion rests on a replay suite of past work items and escaped defects, judged at least as well as by the certified release, and on proof that every guardrail probe is still refused.

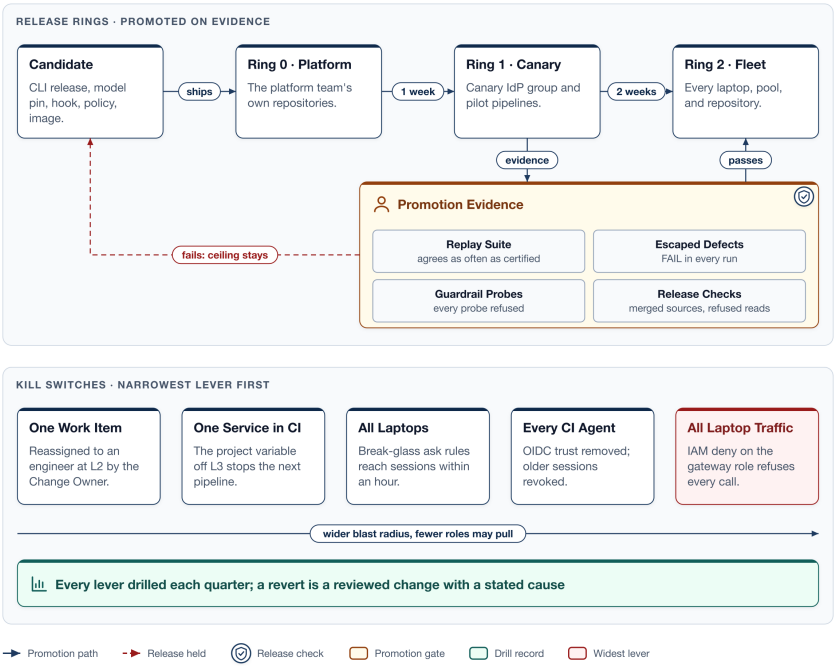

Exhibit 5. Factory changes reach the fleet through rings and are promoted only on replay evidence and release checks, while autonomy drops through graduated kill switches, from one work item to every laptop.

Autonomy Drops Through Graduated Kill Switches, Narrowest Lever First

The rule is to pull the narrowest lever that covers the blast radius and widen only if the signal persists, because every wider lever stops work for people who were not affected.

Scope	Who may pull it	Takes effect	Who restores it
One work item	Change Owner	At once	Change Owner
One service	Platform on-call or Change Owner	Next pipeline or next merge	Change Owner
Every laptop	Platform on-call, AppSec lead	Within an hour	Platform lead
Every pipeline agent	Platform on-call, AppSec lead	Next identity issuance	Platform lead, AppSec consulted
All laptop model traffic	CISO or delegate	Once the cloud deny propagates	CISO or delegate

Every lever is drilled each quarter, and a return to Level 3 is itself a reviewed change with a stated cause.

The Factory Is Operated as a Tier-1 Service

Every agent session depends on the gateway, the verification jobs and the telemetry path, so those paths carry objectives, set by each organization from a quarter of its own measurements:

Objective	Recommended starting target
Gateway sign-in and inference available	99.9% of minutes per month
Spend caps enforced	No unmetered window over 15 minutes
Verification latency	Push to critic verdict within 20 minutes at the 90th percentile
Release checks	Every image and gateway release passes before rollout

When the gateway spends its monthly error budget, releases other than fixes wait.

Eight Risks Decide Whether the Factory Scales

Risk	Control
Business rules lost in ticket threads	Durable specifications; Spec Gate
Agents weakening their own guardrails	Fail-closed checks on protected paths; one workspace per task
Self-certification	An independent critic; four stations
Credentialed agents in pipelines	Short-lived identity; no push rights; a sandbox
Unattributed spend	Gateway sign-in; caps; cost per merged change
Adoption without new roles	Change Owner role; taught skills; waves
Controls that outlive their reasons	Swap-tested building blocks; dated dispensations
A factory upgrade that breaks every team	Rings; a replay suite; graduated kill switches

Leaders Should Answer These Questions Before Funding the Rollout

CTO and VP Engineering

- Do we hold a full quarter of the five DORA metrics for each service?
- Who is the Change Owner for each domain, and who is the deputy?
- How fast can we move one service, or every laptop, from L3 back to L2?

CISO

- Which agent jobs hold standing keys or push rights?
- Which “never” rules exist only as prose?
- Does central policy survive the gateway rollout?
- Which waivers are open, and when does each expire?

CFO and FinOps

- Does telemetry reconcile with each provider bill?
- Is spend attributed by repository and capped per person?
- What does a merged change cost us in agent spend?