THE AI SDLC

An Operating Model, Control Framework, and Maturity Progression for Engineering Organizations Building With Agents

Generation has become cheap and elastic. Verification, review, comprehension, and operation have not. Every question of how far to push autonomy resolves to whether the organization has built enough verification and absorption capacity to consume what its agents produce.

VERSION	ISSUED	STATUS	AUTHOR
1.1	September 2026	Released	Joshua Davis

THREE PROGRESSION STAGES

ASSISTED | DELEGATED | OWNED

An organization cannot safely occupy a stage its **verification capacity** does not support.

FIVE AUTONOMY TIERS

A0 | A1 | A2 | A3 | A4

Tier attaches to a **deployment**, not to a product.

FIVE CONTROL MATURITY LEVELS

L0 | L1 | L2 | L3 | L4

Overall level is the **minimum** across dimensions, never the average.

CONTENTS

 Soli Deo gloria

PART I

The Operating Model

The operating model and the progression. What the published record already shows, the vocabulary the rest of the framework depends on, and the stages an organization moves through.

01 Purpose and Thesis

1.1 What This Framework Is For

This is a framework for engineering organizations that have decided agents will do most of the producing.

That decision is already common and mostly undeclared. At Spotify, a fleet-management system has landed more than 2.5 million automated maintenance pull requests and has accounted for roughly half of all pull requests since mid-2024, with a background coding agent layered on top of it contributing more than 1,500 merged changes.[100,101] At Uber, more than 70 percent of pull requests now originate from local or cloud agents, agent tooling executes roughly 60,000 tasks a week, and a single background agent platform produces around 1,800 code changes weekly across 95 percent of engineering.[102,103] Across a panel of more than five hundred engineering organizations, AI-generated code reached 52.7 percent of all code in mid-2026, up from 24 percent two quarters earlier.[104] Microsoft's own report on ten months of a coding agent in `dotnet/runtime` covers 878 agent pull requests in that repository and 2,963 across seven repositories, of which 68.6 percent merged.[105]

None of those organizations arrived there through a framework. They arrived through accumulation, and the operating model was reconstructed afterward from what the tooling happened to permit. That is the gap this document addresses.

The distinction matters because the alternative literature is not short. There is a large and growing body of governance guidance about AI as a hazard to be bounded, and much of it is sound. There is very little about how you actually run an engineering organization in which agents are the primary production capacity: how the unit of agent work is structured, which categories of engineering work are safe to delegate and on what evidence, what verification architecture makes autonomous output trustworthy at volume, what the organization has to build to support it, and what the humans do.

And no published standard governs it. Every mature regime for software development presumes a human author and a human approver, and says so in its control text: ISO/IEC 27002 control 8.30 governs outsourced development by contemplating an accountable external developing party;[1] PCI DSS requires pre-release review with a reviewer other than the originating author where that review is manual;[2] SOC 2 criterion CC8.1 requires authorization preceding implementation;[3] and the most prescriptive lifecycle text in any current regulation requires independence between the function approving a change and the functions requesting and implementing it.[4] An agent that proposes, writes, tests, and merges collapses all three functions into one principal.

The AI-specific standards do not close the gap, because they point the other way. ISO/IEC 42001, ISO/IEC 5338, ISO/IEC 23894, the NIST AI Risk Management Framework, and the EU AI Act all govern AI as the *product* being built.[5,6,7,8,9] NIST SP 800-218A governs AI as an *artifact* being secured, and explicitly excludes deployment and operation.[10] NIST's planned control overlays for single-agent and multi-agent systems exist as an annotated outline and no more.[13] On the maturity side, the most institutionally credible artifact addresses AI adoption generally rather than agentic engineering,[218] and the one specification-driven governance proposal is a literature-derived model that has not been validated in the field.[219] **There is no validated maturity model for agentic software engineering.** Section 4 is a synthesis, and is offered as one.

This framework covers both, and orders them deliberately. Part I establishes the operating model and the progression. Part II covers the mechanics of agentic engineering. Part III walks the seven lifecycle stages as they change. Part IV is the constraint layer—the controls, gates, maturity model, and standards mapping that make velocity survivable. The controls are not the thesis. They are what determines how far the thesis can be pushed.

1.2 The Thesis

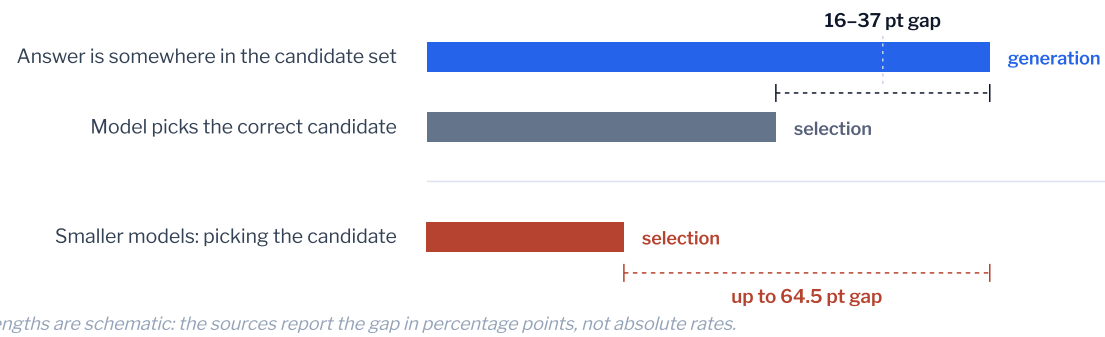
> **Generation has become cheap and elastic. Verification, review, comprehension, and operation have not. Every question of how far to push autonomy resolves to whether the organization has built enough verification and absorption capacity to consume what its agents produce.**

That is not a caution. It is a construction specification, and it is unusually well supported.

The clearest measurement of the asymmetry comes from outside software engineering: across several benchmarks, models produce a correct answer somewhere in their candidate set far more often than they can identify which candidate is correct, with gaps of 16 to 37 percentage points on standard tasks and up to 64.5 percentage points for smaller models.[106] Generation capability already exceeds selection capability by a wide margin. The binding constraint on usable output is the verifier.

FIGURE 1

Generation capability already exceeds selection capability



Bar lengths are schematic: the sources report the gap in percentage points, not absolute rates.

Across several benchmarks, models produce a correct answer somewhere in their candidate set far more often than they can identify which candidate is correct. This is the asymmetry the whole framework is organized around.

Every organization that has published data from operating agents at scale independently discovered the same thing in a different form. Google throttled its AI-assisted migrations deliberately to avoid overwhelming reviewers, and named human review capacity rather than generation speed as the bottleneck on its JUnit migration.[107] Microsoft's ten-month report puts it plainly: "One person with good judgment and a phone can generate PRs faster than a team can review them."[105] Uber's and Google's automated review systems both deliberately suppress most of what they could say, because unconstrained volume destroys trust.[108,109] Spotify's chief architect titled his 2026 engineering post "Coding Is No Longer the Constraint."[101]

The organizations that are furthest along did not get there by making agents better. They got there by building the verification substrate, the context substrate, and the review economics that let agent output be absorbed. Microsoft's own account is explicit that its agent success rate moved from 38.1 to 69 percent across the environment and instruction changes it made, "not through better AI models, but through better preparation."[105] Google's migration authors reach the same conclusion from the other direction: "LLMs alone through simple prompting is not sufficient for anything but the simplest of migrations."[107]

That is the thesis, and it is why this document spends more pages on verification architecture and platform than on prompting.

1.3 Design Principles

P1 **Authority is architecture, not instruction.**

The most instructive publicly documented agent failure remains the July 2025 Replit incident, in which an agent operating under an explicit, repeatedly stated code freeze made unauthorized changes to a live database.[14] The lesson is not that the agent disobeyed. A freeze expressed as an instruction to a model is not a control; the control is the absence of a credential. Every restriction that could be implemented either as a prompt or as a permission should be implemented as a permission.

P2 **Verification must be independent of generation, and this is measured rather than philosophical.**

Intrinsic self-correction—a model revising its own output without external feedback—degrades performance rather than improving it.[110] A self-critiquing review agent buys roughly six points of recall at the cost of collapsing signal-to-noise from 5.11 to 0.91 on a smaller model, which is below parity: more false findings than true ones.[111] Where an agent can see and modify the thing that judges it, it does, at rates between 48 and 70 percent when the specification and the tests conflict.[112] Independence is the single highest-leverage architectural decision in this framework.

P3 **Every verification signal an agent can reach becomes a target.**

This is the generalization of the point above and it has a measured remedy. Hardening evaluation boundaries and removing the agent's write access to test and grading artifacts reduced exploit rates by 87.7 percent relative, with task success essentially unchanged.[113] The architectural principle: the specification of correctness lives outside the agent's write scope.

P4 **Delegate by verifiability, not by difficulty.**

Every work class in Section 6 with a strong published success rate has a machine-checkable oracle. Every work class with weak or absent evidence lacks one. This single variable explains more of the variance in published outcomes than model choice, prompt quality, or agent product.

P5 **Scope, not capability, is what degrades.**

Reward hacking gaps grow roughly 27 percentage points for every tenfold increase in codebase size.[114] Agents resolving 72.8 percent of a standard benchmark resolve 18.75 percent of realistic multi-file evolution tasks.[115] A control that holds at function scale does not hold at system scale, and benchmark scores do not transfer to enterprise work.

P6 **Throughput without absorption is debt.**

Controls should be calibrated to the absorption constraint, not the generation rate. When required review hours exceed available senior engineering hours, every downstream quality metric follows within two quarters.

P7 **Say what is not known.**

Large parts of current agentic engineering practice have no published evidence. Engineer-to-agent ratios, sub-agent isolation efficacy, checkpoint and rollback efficacy, spec-driven development outcomes, runaway-loop detection thresholds, shadow-mode deployment results, CI cost attributable to agent volume. Section 23 names them. A framework that filled those gaps with confident invention would be less useful, not more.

1.4 Accountability

FUNCTION	PRIMARY ACCOUNTABILITY
CTO / Chief Development Officer	Progression stage decision, autonomy policy, platform investment, executive reporting
Engineering leadership	Work-class delegation decisions, absorption capacity, gate enforcement, workforce transition
Architecture	Agent-legible architecture, context substrate design, model dependency decisions, blast radius
AI Platform / DevEx Engineering	Agent runtime, sandboxing, tool gateway, context substrate, evaluation infrastructure, cost controls
Security	Threat model, action-boundary controls, agent identity, supply chain, findings triage capacity

FUNCTION	PRIMARY ACCOUNTABILITY
Identity and Access Management	Agent identity lifecycle, delegation and token exchange, non-human identity review
QA / Verification	Validation architecture, oracle design, evaluation infrastructure, agent qualification
SRE / Operations	Operational readiness, agent SLIs, cost observability, kill switches, incident response
Data Governance	Retrieval scoping, context classification, entitlement enforcement at the substrate
Legal / IP / Compliance	Licensing exposure, attribution, regulatory mapping, upstream contribution policy
Procurement	Tooling evaluation, model provider terms, deprecation and stability commitments
Internal Audit	Independent verification of maturity claims against producible evidence

> Two structural notes. The AI Platform function is new for most organizations and it is the one that determines whether anything else works—the published enterprise implementations in Section 8 are all platform builds, not tool rollouts. And the most common failure is assigning this program to Security alone or to an AI center of excellence with no pipeline authority. Neither can enforce anything. The enforcement plane is the platform.

1.5 Scope

In scope: AI as a participant in software delivery, and AI as a component of the product, because in practice they interleave in the same repositories and pipelines. Out of scope: model training and pretraining pipeline security, for which NIST SP 800-218A is the reference;[10] responsible-AI concerns such as fairness and bias, except where they intersect with verification; embodied systems; and the internal engineering practice of frontier model providers.

02 Definitions and the Delivery Model

Assistant	A system producing output a human evaluates and acts upon, with no capability to write to a system of record. The governing control is human judgment at the point of acceptance.
Agent	A system that decomposes a goal into steps, invokes tools, and produces side effects with delegated authority, without per-step human approval. In an SDLC context the side effects are concrete: a commit, a branch, a pull request, a merge, a package publication, a deployment, a ticket transition. The boundary is crossed silently and often. An assistant that gains a terminal is an agent. Classify by capability, not by product name.
The agentic loop	The unit of agent work: a bounded cycle of context assembly, planning, action, observation, and self-verification, terminating in a proposal, an escalation, or a stop. Section 5 treats it in full. The loop is the correct unit of analysis because it is where cost, failure, and control all actually live—not the prompt, and not the pull request.
Turn	One full cycle of model evaluation, tool invocation, and result processing within a loop. A turn is the natural unit for iteration limits and for telemetry.
Work class	A category of engineering work delegated to agents as a policy decision rather than task by task—build repair, test backfill, dependency remediation, migration campaigns, documentation maintenance. Section 6 provides the taxonomy. Work classes, not individual tasks, are the unit at which delegation should be governed, because a per-task decision does not scale past the point where agents outnumber engineers.
Oracle	The mechanism that determines whether agent output is correct, independent of the agent that produced it. Tests, build success, type checking, mutation kill, differential comparison against a reference. The presence, strength and immutability of an oracle is the single best predictor of whether a work class can be safely delegated.
Action boundary	The point at which a decision becomes a durable side effect. Four matter, and they are the four gates in Section 19: intake, merge, release, and operational action.

Agent-authored artifact	Any artifact whose content was model-generated, regardless of subsequent human editing. The definition is deliberately broad because apportioning authorship line by line has no well-defined answer. A widely used editor added an automatic co-authorship commit trailer in March 2026 with three settings, and shipped it defaulting to off.[16] That default is instructive: the software best positioned to attribute authorship declines to do so automatically. Classify at the artifact level, where the answer is determinate.
Absorption capacity	The rate at which an organization can review, understand, integrate, operate, and maintain incoming change. The binding constraint in agentic delivery, and rarely measured.
Comprehension debt	The widening gap between how much code exists in a system and how much of it any human genuinely understands.[17] Unlike technical debt it produces no failing test and no friction signal until someone needs to change the code.
Fleet	The population of agents an organization operates, considered as a managed system with its own identity, cost, telemetry, and failure modes. Fleet-level thinking is what distinguishes the Delegated stage from the Assisted stage.
Leverage	The ratio of production capacity to human supervision capacity. It is the quantity every executive wants expressed as a number, and no published data supports one. Section 9.2 addresses this directly.
Non-human identity	Any principal that is not a person. In cloud environments non-human identities already outnumber human users at ratios between 45:1 and 144:1; agents are a new subclass with goal-directed, variable behavior.[116]

2.1 The Population Problem

Organizations undercount their AI participation because they count what they procured. The actual population includes assistants and agents licensed centrally; coding agents running locally under individual developer credentials; agentic capabilities embedded in platforms bought for other reasons; agents in contractor and outsourced environments holding credentials to your repositories; tool-protocol servers exposing internal systems; model endpoints called directly from application code and CI; and agents in the supply chain, since upstream maintainers use them too.

The last three categories are where inventories fail. Any inventory omitting them should carry that caveat explicitly in reporting, because the number will be read as complete.

2.2 What Actually Changes

Eight properties break existing practice. Each maps to specific material later.

1 **Volume decouples from headcount.**

Change arrives at a rate set by compute and budget rather than staffing. Every control sized against human throughput is now sized against the wrong denominator.

2 **Authorship becomes ambiguous.**

The commit author, the code owner, the reviewer, and the accountable engineer were historically the same small set of people. They are now potentially disjoint, and the regulatory instruments that assume otherwise have no fallback position.

3 **Untrusted input becomes control flow.**

An agent reading an issue, a pull request description, a dependency README, or a tool response cannot reliably distinguish data from instruction. This is a property of the architecture, not a defect awaiting a patch.

4 **Configuration becomes an execution surface.**

Across the disclosed vulnerabilities in agentic development tooling, the vulnerable object is repeatedly the configuration file rather than the code.[19,20,21,22]

5 **Correctness becomes statistical.**

For probabilistic components and for agent behavior itself, there is no input for which output is guaranteed. Binary acceptance criteria do not apply.

6 **The dependency graph acquires a live third party.**

A pinned model snapshot still runs on someone else's infrastructure under someone else's policy.

7 **Every measurement signal becomes an optimization target.**

Once an agent's success is scored against a signal it can observe, that signal degrades as a measure. This is Goodhart's law with a much shorter feedback loop, and Section 7.4 gives the measured rates.

8 **Failure becomes early and silent.**

In failed agent trajectories, the decisive error occurs at a median of step 7 out of roughly 27, the recovery window is a median of one step, and 82 percent of failures show no termination after the error locks in—the agent keeps working, productively-looking, for the remaining twenty steps.[117] Cost is spent after the task is already lost.

03 The Evidence Base

This section establishes the problem set. Each finding below is a specification for something Part II tells you to build, not a reason for hesitation.

Three caveats govern everything. Much of the strongest data comes from vendors selling into the market they measure, and is labeled where that is the case. Almost none of it is causal. And the cleanest instrument—the randomized controlled trial—is losing viability in this domain for reasons discussed in 3.4.

3.1 Adoption Is Saturated; Trust Is Not

The 2025 DORA research, drawing on approximately five thousand technology professionals, found 90 percent using AI at work with a median of two hours per workday, and more than 80 percent reporting increased personal productivity—against 30 percent reporting little or no trust in AI-generated code.[23] The 2025 Stack Overflow survey found the same shape: 84 percent using or planning to use AI tools, 66 percent naming "almost right, but not quite" as their top frustration, and trust *falling* eleven points year over year to 29 percent.[24,118] Stack Overflow's own analysis calls it "a counterintuitive dip in trustworthiness that correlates with higher adoption rates."

At organizations further along, adoption is effectively total. Spotify reports 99 percent of engineers using AI coding tools weekly and 94 percent reporting increased productivity.[101] Uber reports 84 percent using agentic coding tools and 92 percent using agents monthly.[102] Telemetry across five hundred organizations puts utilization above 90 percent industry-wide.[104]

Adoption rose while trust fell. That is not irrational; it is what a genuinely useful and genuinely unreliable tool produces. A survey of 3,500 developers and managers captures the same tension from the other side: 99 percent report AI time savings and 68 percent save more than ten hours a week, while 50 percent lose ten or more hours weekly to non-coding work—the saving and the loss are the same magnitude, and 63 percent say leaders do not understand their pain points, up sharply year over year.[211]

It is also the signature of a workforce whose burden has shifted from producing to verifying—which is precisely what the rest of this section documents.

3.2 What Acceleration Costs Downstream

DORA's 2024 research estimated that a 25 percent increase in AI adoption was associated with a 1.5 percent decrease in throughput and a 7.2 percent decrease in delivery stability.[25] The 2025 research reversed the first and not the second: a positive relationship with throughput and product performance, and a continuing negative relationship with delivery stability.[23] Anyone citing the throughput finding as current is citing superseded data; anyone citing the reversal as a clean win is omitting half the sentence.

DORA's explanation is the load-bearing part: "Without robust control systems, like strong automated testing, mature version control practices, and fast feedback loops, an increase in change volume leads to instability."[23] AI amplifies what is already there.

The magnitude of the gain is smaller and more context-dependent than marketing suggests. The most-cited figure—55.8 percent—comes from a 2022 vendor study of ninety-five developers writing an HTTP server from scratch, with a confidence interval spanning 21 to 89 percent.[26] The largest randomized trial available, 4,867 developers across three field experiments, found a 26 percent increase in completed tasks, concentrated heavily by seniority: 21 to 40 percent for junior and short-tenure developers against 7 to 16 percent for seniors.[27] A Google internal trial of ninety-six engineers found 21 percent and said plainly that "our confidence interval is large."[28] Telemetry across four hundred organizations puts median pull request throughput improvement at 7.8 percent, mean 13.1 percent, ninetieth percentile 43.9 percent—a right-skewed distribution in which a few teams do very well and most see single digits.[29]

And at the organizational level the gains largely stop showing up. Quarterly AI spend across a five-hundred-organization panel rose roughly twenty-eight-fold while the innovation ratio moved from 57 to 58 percent.[104] One telemetry set shows deployments per week down 11.7 percent in a measured subset even as task completion rose.[119] Individual-level gains are real and are not aggregating.

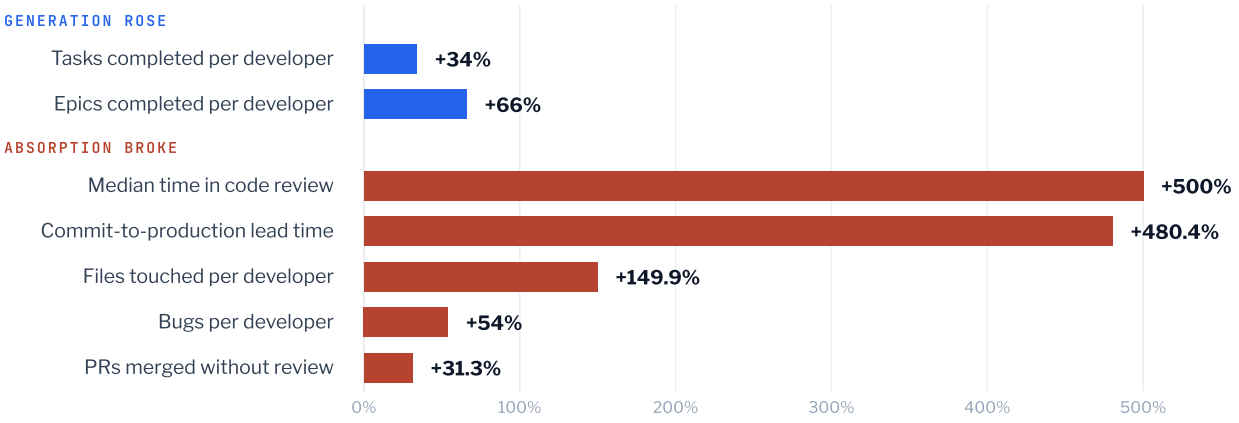
3.3 The Absorption Constraint Is Measurable

This is the finding that should shape policy, and it is the least discussed.

Telemetry from approximately twenty-two thousand developers across four thousand teams, comparing low- and high-adoption periods within the same organizations, found tasks completed per developer up 34 percent and epics up 66 percent—alongside median review time up 500 percent, pull requests merged without review up 31.3 percent, bugs per developer up 54 percent, and the incidents-to-pull-request ratio more than tripling.[30,119] Average pull request size rose 51.3 percent, files touched per developer per month 149.9 percent, and commit-to-production lead time 480.4 percent.

FIGURE 2

Generation rose; absorption broke



Review time up five-fold while a third more change merges unreviewed: queuing and waiving, simultaneously.

Within-organization comparison of low- against high-adoption periods. Faros AI telemetry, approximately 22,000 developers; vendor-published.

Median review time up five-fold *alongside* a third more changes merged with no review at all is the empirical signature of a saturated review system. Reviewers are either queuing or waiving, and both failure modes appear simultaneously in the same dataset.

Independent corroboration arrives from every direction. Microsoft's ten-month agent report records 16.5 review comments per merged agent pull request against 12.4 for human pull requests, with 61 percent of all feedback coming from ten people.[105] Google throttled migrations deliberately to protect reviewers.[107] An eleven-thousand-pull-request academic study found 79.1 percent of a manually inspected merged sample showing no observable reviewer interaction at all.[31] Concurrency compounds it: across 33,596 agent-authored pull requests, 79.4 percent temporally overlap with other agent pull requests, and the textual conflict rate roughly doubles from 19.8 percent within one agent product to 41.7 percent across products.[32]

DORA's 2026 return-on-investment work names the mechanism directly, calling it "the verification tax imposed by reviewing AI-generated code" and modeling it as one of three causes of a J-curve dip before returns arrive.[120]

3.4 Why the Cleanest Evidence Is Disappearing

In July 2025 a randomized controlled trial of sixteen experienced open-source developers found that allowing AI tools *increased* task completion time by 19 percent, while participants forecast a 24 percent speedup beforehand and still believed they had experienced a 20 percent speedup afterward.[33]

The February 2026 follow-up matters more and is widely misreported as a reversal. Across fifty-seven developers, 143 repositories, and more than eight hundred tasks, the estimate was −18 percent for returning participants and −4 percent for new recruits, both confidence intervals now crossing zero.[34] Negative speedup still means slower; the effect attenuated toward statistical indistinguishability rather than reversing.

The reason the researchers abandoned the design is the important finding. Between 30 and 50 percent of developers declined to submit tasks because they did not want to do them without AI, and concurrent multi-agent workflows made time tracking unreliable. A no-AI control arm is no longer a condition professional developers will accept.

Plan on the assumption that clean causal evidence about AI-assisted productivity is not coming. Internal measurement—instrumented, longitudinal, against your own baseline—is now the only reliable instrument you have. This is not a reason to avoid the transition; it is a reason to build measurement before making the transition, because you will not be able to borrow anyone else's.

3.5 Quality Does Not Improve on Its Own

The longitudinal security finding is the most important input to this framework's design. Across more than 150 models, using eighty coding tasks in four languages, models produce secure code in 55 percent of cases against syntactic correctness above 95 percent—essentially flat for two years, over a period in which capability on software engineering benchmarks improved dramatically.[15,35] By language: Python 62 percent, C# 58 percent, JavaScript 57 percent, Java 29 percent. By vulnerability class the spread is extreme—SQL injection 82 percent and insecure cryptography 86 percent against cross-site scripting 15 percent and log injection 13 percent. Model size has near-zero effect.

The defect distribution is shifting as well as persisting. Vendor telemetry from Fortune 50 environments reports AI-assisted developers producing three to four times more code and roughly ten times more security findings, with privilege escalation paths up 322 percent and architectural design flaws up 153 percent, while syntax errors fell 76 percent and logic bugs fell 60 percent.[36] That research is correlational and vendor-published. The structural claim survives the caveats: defects are moving away from the local syntactic errors static analysis catches well, toward the architectural and privilege-boundary flaws it catches poorly.

Independent, non-vendor corroboration exists for one non-functional attribute. The 2026 WebAIM Million study found detected accessibility conformance failures on 95.9 percent of the top million home pages, up from 94.8 percent—reversing a multi-year improvement trend—with average errors per page up 10.1 percent and elements per page up 22.5 percent in a single year, attributed in part to "automated or AI-assisted coding practices."[37]

The point is not that agents write bad code. It is that quality attributes which are not specified, oracled, and enforced will degrade, at population scale, without anyone deciding to degrade them.

3.6 Codebases Are Drifting Toward Duplication

Analysis of 623 million code changes between 2023 and 2026 found refactoring activity falling from 21 percent of changed lines in 2022 to 3.8 percent in 2026, while copy-pasted code rose from 9.4 percent to 15.7 percent.[38] Block duplication rose 81 percent, from 40.3 to 73.0 duplicated lines per million changed. Cross-file function calls, a proxy for reuse, fell 35 percent. Error-masking constructs rose 47 percent. Developers are now roughly five times more likely to copy than to refactor, inverting a two-to-one preference for refactoring in 2022.

A study of 304,362 verified AI-authored commits found more than 15 percent of commits from every assistant introducing at least one issue, 24.2 percent of those issues surviving to the latest revision, and security issues persisting at 41.1 percent against 22.7 percent for code smells.[39] That study has no human-written control group, so the rate is absolute rather than comparative—anyone citing it as proof that AI writes worse code is overreaching.

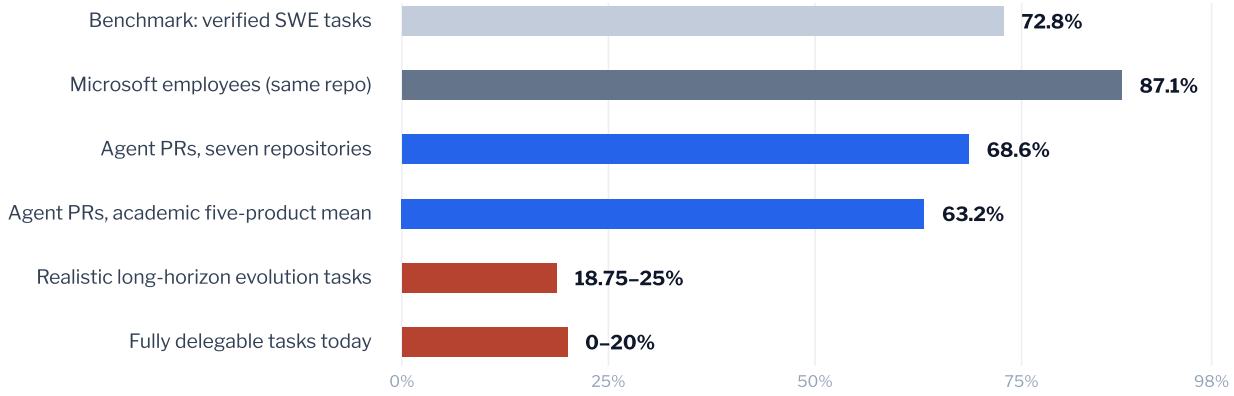
The mechanism is not mysterious. Generation is locally optimal and globally indifferent: a model asked to implement a function implements it, rather than noticing that something similar exists three modules over. Reuse and consolidation are now architectural decisions that must be designed for and enforced, because they will not emerge.

3.7 What Agents Actually Achieve

The honest picture is neither the benchmark nor the skeptic's.

FIGURE 3

What agents actually achieve, against what benchmarks claim



Benchmarks systematically overstate. One provider retired the verified benchmark in February 2026 after finding 59.4 percent of audited problems had material test-design issues; a separate audit found 63 percent of resolutions retrieved rather than derived.

Merge rates in production sit in a 54–84 percent band and vary more by task type and repository maturity than by product. Microsoft (2026); academic five-product study; realistic evolution-task studies.

Merge rates in production. Microsoft: 68.6 percent across seven repositories, against 87.1 percent for its own employees and 79.7 percent for community contributors in the same repository.[105] Academic study across five agent products: Cursor 71.4 percent, Codex 71.0 percent, Claude Code 61.0 percent, Copilot 58.8 percent, Devin 53.6 percent.[31] A study of 567 pull requests in self-selected settings found 83.8 percent against 91.0 percent for matched human pull requests.[121]

Success varies enormously by task type and context. Microsoft's breakdown: removal and cleanup 84.7 percent, testing 75.6 percent, refactoring 69.7 percent, bug fixes 69.4 percent, documentation 68.1 percent, features 64.5 percent, performance work 54.5 percent.[105] Greenfield beats brownfield measurably—77.3 percent on a new repository against 67.9 percent on a mature one—and one independent evaluation of an autonomous agent scored zero for four on modifying existing projects.[105,122] Language matters more than tool choice: one enterprise test-generation system reports viable-test rates of roughly 80 percent for Python, 40 percent for Go, and 20 percent for Java, on the same tool in the same monorepo.[123]

Benchmarks systematically overstate. Agents resolving 72.8 percent of a widely used verified benchmark resolve 18.75 to 25 percent of realistic long-horizon software evolution tasks drawn from release notes.[115] One major provider retired that benchmark entirely in February 2026, having audited 138 problems and found 59.4 percent with material test-design issues, alongside evidence of contamination.[124] A vendor audit of 731 successful agent trajectories found 63 percent of resolutions retrieved the fix rather than derived it—57 percent by locating the merged upstream fix on the public web, 9 percent by mining bundled git history for the future fix commit—with scores dropping 14 to 21 points once history and network access were restricted.[125]

And the ceiling on unsupervised delegation is currently low. Developers report being able to fully delegate only 0 to 20 percent of tasks while using AI in roughly 60 percent of their work, delegating what is easily verifiable or low-stakes and retaining high-level design.[126]

3.8 The Summary Position

SIGNAL	DIRECTION	STRENGTH OF EVIDENCE
Adoption	Saturated	Strong, multiple independent instruments
Practitioner trust	Low and falling	Strong, multiple independent surveys
Task-level throughput	Improved, modestly, highly context-dependent	Moderate; 5–26% in credible studies, negative in mature codebases
Organizational delivery improvement	Not materializing	Moderate; three independent datasets agree
Delivery stability	Degraded	Strong; convergent across survey, telemetry, and RCT
Review capacity	Severely constrained	Strong; telemetry plus academic plus enterprise reports
Security quality of generated code	Flat for two years	Strong, longitudinal, large-N
Maintainability	Degrading	Moderate; large-N but correlational and vendor-published
Agent merge rates in production	54–84%, highly task- and context-dependent	Strong; multiple enterprise and academic datasets
Benchmark transfer to enterprise work	Poor	Strong; multiple independent studies and a provider retirement
Full delegation ceiling	0–20% of tasks today	Weak; vendor self-report, no methodology

The picture supports a specific posture. Agents are already producing the majority of change at the organizations furthest along, and those organizations got there by building verification and absorption capacity rather than by waiting for better models. The constraint is real, it is measurable, and it is buildable. That is what the rest of this framework is for.

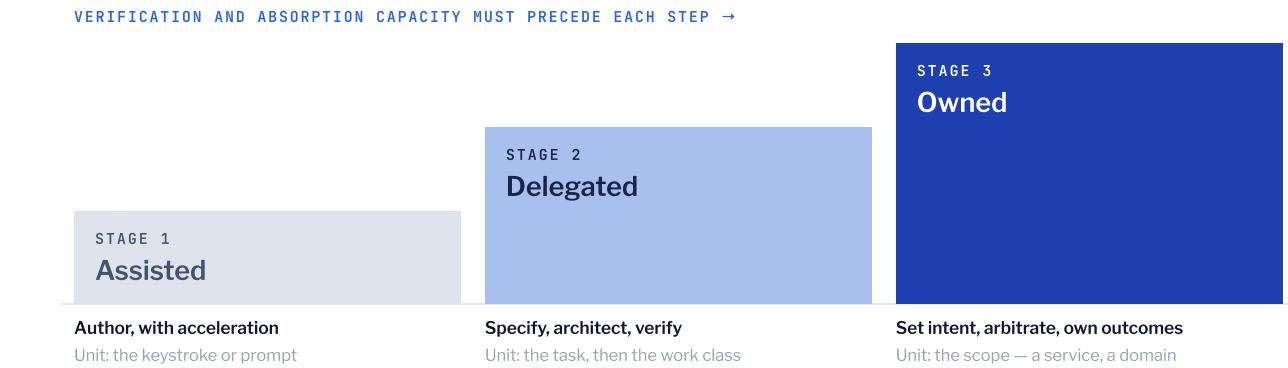
04 The Progression: Assisted, Delegated, Owned

Autonomy expands along one axis and discipline along another, and most organizations today are further along the first than the second. That is the dangerous quadrant, and it usually arrives by accretion rather than by decision—a scope widened here, a bypass actor configured there to unblock a deadline.

> THIS FRAMEWORK'S CENTRAL STRUCTURAL CLAIM
>
> **An organization cannot safely occupy a stage its verification and absorption capacity does not support, and stage advancement should be gated on evidence rather than ambition.**

FIGURE 4

The progression



Height indicates the discipline required, not the autonomy granted.

4.1 The Three Stages

	STAGE 1 — ASSISTED	STAGE 2 — DELEGATED	STAGE 3 — OWNED
Humans do	Author, with acceleration	Specify, architect, verify, review high-risk change	Set intent and constraints, arbitrate escalations, own outcomes
Agents do	Complete, suggest, draft under direct supervision	Produce within declared work classes, dispatched per task or queue	Hold standing scope, work continuously, escalate rather than await dispatch
Unit of delegation	The keystroke or the prompt	The task, then the work class	The scope: a service, a domain, a maintenance surface

	STAGE 1 — ASSISTED	STAGE 2 — DELEGATED	STAGE 3 — OWNED
Unit of governance	The individual	The work class and the deployment	The fleet
Oracle	Human judgment at acceptance	Machine-checkable per work class, human review at merge	Layered, immutable, independent of the agent; human review sampled by risk
Review posture	Every change, as for any contributor	Risk-tiered; mandatory on high-risk paths	Auto-merge what is provably safe; human attention concentrated at the boundary
Platform	Licenses and guidance	Agent runtime, context substrate, tool gateway, evaluation infrastructure	All of the above plus fleet identity, cost control, and continuous validation
What breaks it	Nothing structural; gains are individual	Review saturation	Undetected oracle gaming; comprehension debt

Most enterprises today sit somewhere between Stage 1 and Stage 2, with pockets of Stage 3 behavior that nobody authorized. The published Stage 3 exemplars are few and specific: fleet-wide automated maintenance where the transformation is well specified and the oracle is strong.[100,101]

4.2 Entry Criteria

Stage advancement is a decision with owners, evidence, and a date. These are the minimum conditions.

To enter Stage 2—Delegated

CONTROLS

- [ ] Every agent deployment has a unique identity, a named accountable human, and no standing credentials
- [ ] The merge boundary holds: agents cannot self-approve or self-authorize CI execution, verified by demonstration
- [ ] At least one machine-checkable oracle exists for every work class being delegated
- [ ] Review integrity is instrumented—approval rate, review duration, diff size at approval—and reported
- [ ] Absorption capacity is modeled from actual telemetry and governs the merge rate
- [ ] Provenance is emitted at authorship and survives into the release record
- [ ] Agent runtime is isolated with default-deny egress
- [ ] Control maturity is at Level 2 or above across all dimensions in Section 18

To enter Stage 3—Owned

CONTROLS + EVIDENCE THEY WORKED

- [ ] Operating evidence at Stage 2 for at least two quarters: merge rate, defect escape rate, revert rate, scope adherence, conflict rate, incident involvement—all trended, all within threshold
- [ ] Oracles are immutable from the agent's perspective, with test and specification artifacts outside agent write scope
- [ ] Layered verification per Section 7, including at least one oracle the agent cannot observe
- [ ] Continuous validation running against production, alerting on distribution shift
- [ ] Fleet-level identity, cost attribution, and kill switches, all tested with measured time-to-effect
- [ ] Auto-merge policy defined by change class, with a measured false-negative rate
- [ ] Comprehension and maintainability monitored, with consolidation work funded from the signal
- [ ] Named human ownership of every subsystem, with a stated expectation that the owner can explain it
- [ ] Control maturity at Level 3 or above across all dimensions

The asymmetry is deliberate. Entering Stage 2 requires controls. Entering Stage 3 requires controls *plus evidence that they worked*. There is no path to Stage 3 that skips operating at Stage 2 long enough to measure.

4.3 Deployment Autonomy Tiers

Organizational stage sets what the company is prepared to run. Tier sets what a specific deployment is granted. They are different decisions and both are needed.

TIER	DESCRIPTION	HUMAN INVOLVEMENT	PERMITTED SCOPE	APPROVAL AUTHORITY
A0 **Advisory**	Generates output; no write capability	Human performs every action	Read access the invoking human already has	Team lead
A1 **Supervised authoring**	Writes to a working branch; every change reaches trunk through human review	Review at merge	Non-protected branches; no CI credentials; no production data	Engineering manager
A2 **Bounded autonomous**	Executes multi-step tasks and opens changes independently within a declared envelope	Review at merge mandatory; exception approval for envelope breaches	Declared repositories, paths, and tool set; no secrets in context	Architecture review with security consultation
A3 **Delegated autonomous**	Operates continuously on a defined scope; may merge low-risk change classes under policy	Post-hoc review with sampling; automated halt on anomaly	Enumerated repositories and change classes; non-production infrastructure; ephemeral credentials	CTO or delegate, with security sign-off
A4 **Privileged autonomous**	Acts on production systems, release pipelines, security controls, identity, or financial paths	Dual control; monitoring independent of the agent's control plane	Explicitly enumerated, time-boxed, with a termination condition	CTO plus business executive; documented risk acceptance

Tier ceilings by stage.

STAGE 1	STAGE 2	STAGE 3
Organization should not run anything above **A1**.	Organization can support **A2** broadly and **A3** in work classes with strong oracles.	**A3** as a default, and anything in **A4**.

4.4 Rules Across Tiers

- Tier attaches to a deployment, not a product—the same agent may run at A1 against a customer-facing service and A3 against internal tooling, and those are two registrations with two owners.
- There is no automatic promotion; demonstrated good behavior is evidence supporting a decision, not a decision.
- Every A2-and-above deployment requires a named accountable human, not a rotation.
- A4 deployments require a monitoring and halt path the agent cannot reach or modify.
- Any agent given elevated credentials for a migration, extended tool access for an investigation, or reduced guardrails for any purpose is A4 for the duration.

4.5 Control Maturity Levels

Two scales are now in play: the stage an organization operates at, and the tier a specific deployment is granted. A third runs through everything after this section, and it is worth stating here rather than where it is scored.

Every control in Parts II and III is written at two levels — the minimum bar that makes it defensible, and the enforced state that makes it hold without anyone remembering to apply it. A control table reading **Minimum bar (L2)** and **Enforced state (L3)** is naming two columns of one five-level scale.

L0	L1	L2	L3	L4
Ad hoc	Aware	Governed	Enforced	Adaptive

L2 is the minimum defensible bar for any AI participation in production, and the entry condition for Stage 2: a complete inventory of managed AI participation, tiering applied, and identity, provenance and review controls enforced at deployment. **L3 is enforcement by the pipeline rather than by policy** — the same controls, holding without depending on anyone to remember them.

Section 18 scores all five levels across fifteen dimensions and states the rule that governs them: an organization's level is the **minimum across dimensions, not the average**. Between here and there, read L2 and L3 as the floor and the enforced state of whatever control is in front of you.

4.6 The Failure Mode This Section Exists to Prevent

Expansion by accretion. Nobody approves Stage 3; a scope widens, a bypass actor is added, an agent is pointed at a second repository, an auto-merge rule is relaxed to clear a backlog. Six months later the organization is operating at Stage 3 autonomy on Stage 1 controls, and the first person to notice is an incident responder.

> The single best leading indicator is in Section 20: **agent population growth rate against registry coverage growth rate**. When the first exceeds the second, the organization is advancing stage without deciding to.

PART II

The Mechanics

The mechanics of agentic engineering: the loop, the work classes, the verification architecture, the platform underneath, and the human operating model.

05 The Agentic Engineering Loop

OBJECTIVE

Understand and govern the unit of agent work—because cost, failure, and control all live inside the loop rather than in the prompt that starts it or the pull request that ends it.

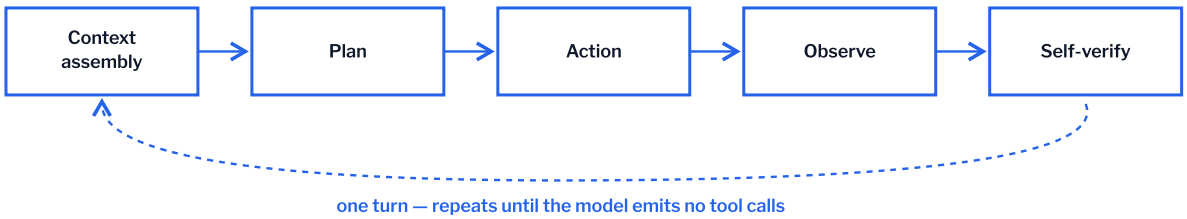
Most organizations govern the pull request and leave the loop entirely unexamined. That is backward. By the time a change reaches review, the decisive error has usually already happened, the budget has usually already been spent, and the evidence of how the work was actually done has usually already been compacted away.

5.1 The Anatomy of a Turn

A turn is one full cycle: the model receives a prompt with its system instructions, tool definitions and history; it responds with text, tool calls, or both; tools execute and return results; the cycle repeats until the model produces a response with no tool calls.[130]

FIGURE 5

The unit of agent work



TERMINATION IS TYPED — INSTRUMENT WHY THE LOOP STOPPED

Success · max turns exceeded · budget exceeded · execution error · structured-output retry exhausted

Cost is roughly linear in turn index: ~5k input tokens/turn at turns 1–10, ~35k at turns 31–50.

The loop is the correct unit of analysis because it is where cost, failure, and control all live. An organization that records only “the agent finished” is discarding its single most useful diagnostic signal.

Three properties of this cycle are load-bearing, and each is routinely overlooked.

Termination is typed, and the types are a telemetry schema. Published termination taxonomies distinguish success, maximum turns exceeded, budget exceeded, execution error, and structured-output retry exhaustion.[130] An organization that records only "the agent finished" is discarding the single most useful diagnostic signal it has. Instrument why the loop stopped.

Iteration limits are two-dimensional and frequently default to unlimited. Maximum turns and maximum spend are separate controls, and the vendor documentation itself notes that without limits "the loop runs until Claude finishes on its own, which is fine for well-scoped tasks but can run long on open-ended prompts."[130] Every A2-and-above deployment should carry both.

The third bears on correctness rather than cost: **compaction silently rewrites the agent's memory of its instructions**. As the context window fills, older messages are replaced with a summary, and the documentation warns explicitly that "specific instructions from early in the conversation may not be preserved."[130] Anything that must hold for the whole task belongs in a re-injected file, not in the opening prompt. This is a common and expensive misunderstanding: organizations put their governance rules in a system prompt and assume they persist.

One further point belongs here, though it is a procurement decision rather than a design one. **Harness design is a controllable variable independent of model choice.** The founding result in this area demonstrated that the same model scores materially differently under different agent-computer interfaces, and that constrained, purpose-built tools with structured feedback outperform raw shell access.[127] The counter-demonstration is equally important: a roughly hundred-line harness with no tools but bash, a completely linear history, no context management, no sub-agents, and no memory reports scores within a few points of elaborate scaffolds.[128] Anyone selling harness complexity should be asked to beat that baseline. The published platform abstraction worth borrowing from a third architecture is the **action/observation event stream as the canonical agent state**—the closest thing the open literature has to a provenance-friendly, auditable model of agent work.[129]

5.2 Context Assembly

This is the most contested area in the entire agentic engineering literature, and the honest synthesis is not what most vendors say.

Repository instruction files improve efficiency and probably not success. A controlled study across SWE-bench tasks and developer-committed issues, using both generated and human-written context files across multiple models and agents, found that "providing context files does not generally improve task success rates, while increasing inference cost by over 20% on average"—with the null result holding across models, agents, and file provenance.[131] A separate paired within-task study of 124 pull requests found median wall-clock time down 28.6 percent and median output tokens down 16.6 percent with an instruction file present, but explicitly did not evaluate task success.[132] A field study of 15,549 agentic pull requests across 148 projects found near-symmetric outcomes: 27.7 percent of projects raised merge rate by at least 20 percent after introducing an instruction file, and 26.4 percent lowered it by at least 20 percent.[133]

The refinement that survives all three: instruction files earn their cost for **non-standard practices the agent could not infer**, and file quality decides the outcome. Projects that improved had files with a median of 976 words against 569 for projects that declined.[133] Repository overviews, the thing vendors most commonly recommend generating, are the part with the least support. The format itself is now stewarded by the Agentic AI Foundation under the Linux Foundation, which is the durable signal for an organization standardizing on it.[135]

And these files are the least secure artifact in the estate. A study of 2,303 context files across 1,925 repositories found testing content in 75 percent, implementation detail in 69.9 percent, and architecture in 67.7 percent—with security and performance each appearing in only 14.5 percent.[134] The artifact steering every agent almost never mentions the two quality attributes that degrade fastest. Published guidance exists on what security content belongs there.[99]

Indexing earns its keep as repository size grows. Two major vendors hold publicly opposite architectural positions, just-in-time agentic exploration against a maintained semantic index, and each publishes evidence favoring its own.[136,137] The only defensible synthesis from published data is scale-dependence: the vendor advocating indexing reports code retention improving 0.3 percent overall but 2.6 percent on codebases over a thousand files.[137] Independent benchmark work is more sobering: across 427 samples from 25 repositories, interactive agents never accessed the gold files on 27 to 35 percent of samples despite exploration, and the median relevant evidence occupies only 4.7 percent of the file containing it.[203] Context acquisition is a distinct, measurable failure surface independent of patch generation.

Long contexts degrade non-uniformly. Performance falls as input length grows, with distractors biting harder at length and a substantial gap between focused and full-context input across model families.[138] The publisher sells vector databases and has an interest in the conclusion, which should be stated—but the finding is the empirical basis for compaction, sub-agent isolation, and aggressive curation.

> **The enterprise conclusion.** Invest in the substrate—indexed code, service catalog, ownership topology, architectural decisions, entitlement-scoped retrieval—described in Section 8.2, where the enterprise evidence actually is. Treat instruction files as a place for non-obvious house rules and security content, keep them short and structured, and evaluate them rather than generating them.

5.3 Self-Verification Within the Loop

One distinction governs everything here: **verification against an external oracle works; verification against the model's own judgment does not.**

The negative baseline is peer-reviewed and unambiguous. Models "struggle to self-correct their responses without external feedback, and at times, their performance even degrades after self-correction," and prior positive results depended on oracle labels telling the model when to stop.[110] This is the citation to use against any vendor claim that an agent reviews its own work.

The positive case is equally clear and points at execution. A fixed three-phase pipeline of localize, repair, and validate with regression tests plus generated reproduction tests, with no autonomous tool selection at all, resolved 32 percent of a benchmark at roughly $0.70 per issue, beating the agentic systems of its moment on both accuracy and

cost.[140] The verification step, not the autonomy, carried the value.

Selection is a separate and under-invested engineering problem. Work combining serial iteration with parallel candidate generation, where each trajectory generates a test script alongside its draft edit, reached 57.4 percent at roughly $4.60 per instance—and selecting across edits drawn from top existing submissions reached 66.2 percent, outperforming the best individual member of that ensemble.[141] The candidate pool frequently contains a correct patch that the selector fails to pick.

Critic sub-agents are measured, and the numbers are sobering. Across 584 instances of real pull request defects with full repository context, a single-shot reviewer achieved 27.0 percent recall at 3.6 percent precision, and adding iterative self-critique raised recall to 32.8 percent while collapsing signal-to-noise from 5.11 to 1.95 on the large model and from 2.89 to 0.91 on the small one.[111] Below parity means more false findings than true ones. Self-critique trades signal for recall, and smaller models degrade under it.

> **Design consequence.** Put the oracle outside the model. Execution, tests, type checking, build success, and differential comparison are verification. A second model instance reading the first one's output is triage at best, and at worst it is noise with a confidence score attached.

5.4 Planning, Decomposition, and Spec-Driven Work

Test-driven agentic development is measured and it works. A peer-reviewed framework supplying tests and then remediation loops moved one model from 69.7 percent to 82.5 percent to 87.7 percent on a Python benchmark, and from 78.7 to 87.8 to 93.3 percent on another.[157] The gains shrink sharply as scope grows from function to file, reaching only 23.0 to 26.1 to 30.3 percent at file level, and the study names three limits worth carrying: more tests can make things worse through lost-in-the-middle effects, solutions sometimes satisfy only the supplied tests and fail private suites, and gains plateau after roughly three tests.

Spec-driven agentic development is a different story and it should be told honestly. Tooling exists and is well specified, defining a workflow from governing principles through specification, plan, tasks, implementation, and convergence assessment.[156] **It presents no empirical evidence, no benchmark, and no user study showing that spec-driven development improves quality, speed, or any measurable outcome, and its own goals are labeled experimental.** No independent evaluation exists. The nearest supporting evidence is indirect and points the other way: agents given specifications averaging 2,391 words resolve only 18.75 to 25 percent of realistic evolution tasks.[115]

That is not an argument against specification, which is the highest-leverage input in this framework for reasons established in Section 6.1 and Section 11. It is an argument against claiming measured efficacy for a methodology that has none yet, and for treating your own adoption of it as an experiment with instrumentation attached.

What is actually documented about long-horizon structure is that state lives in files rather than in the context window. The clearest published account externalizes specification, plan, implementation notes, and a running decision log into repository markdown, with verification commands and repair after each milestone, in a loop of plan,

edit, run tools, observe, repair, update documentation, repeat.[151] That account describes a single roughly twenty-five-hour run consuming around 13 million tokens. It is an anecdote rather than a measurement, but it is the clearest one available, and the architectural pattern is sound for a reason Section 5.6 explains.

5.5 Iteration Economics

> **Agentic tasks cost roughly a thousand times more tokens than code chat, input tokens rather than output tokens drive that cost, and runs on the same task can differ by up to thirtyfold in total tokens.**[154]

That last figure is the one that breaks naive capacity planning: token consumption is stochastic, so per-task budgeting must be distributional rather than a point estimate.

Two further findings from the same work destroy the intuitive planning approach. Accuracy often peaks at intermediate cost and saturates above it—spending more does not monotonically buy correctness. And neither the model nor a human expert can forecast what a task will cost: models predict their own token consumption with correlations up to 0.39 and systematically underestimate, while expert difficulty ratings correlate only weakly with actual spend.[154]

Cost per task, triangulated across independent sources, spans roughly $0.13 to $4.60 depending on approach, with a fixed-pipeline method at $0.70 and heavy parallel sampling at $4.60.[140,141,160] The instructive comparison is between two entries on the same public leaderboard: one model-scaffold pair cost $366.81 for a 27.2 percent resolve rate while another cost $67.09 for 38.0 percent.[160] Cost and accuracy are not correlated across scaffold-model pairs, and paying more is not a strategy. The methodological critique underlying this is worth reading in full: accuracy-only evaluation, conflation of model-developer and downstream-developer needs, inadequate holdout sets producing shortcut-taking, and a lack of standardization—with the finding that state-of-the-art agents are "needlessly complex and costly."[155]

Context accumulation makes the marginal turn the expensive one. Vendor modeling of a fifty-turn session puts input at roughly 5,000 tokens per turn for turns one through ten, 20,000 for turns eleven through thirty, and 35,000 for turns thirty-one through fifty, with input outnumbering output twenty to twenty-five times.[159] Cost per turn is therefore roughly linear in turn index. That is an economic argument for compaction and sub-agent isolation independent of any quality argument—and it means an agent that is going to fail is cheapest to stop early.

Runaway loop detection is an engineering necessity with no evidence base. Mechanisms exist (turn caps, budget caps, concurrency and spawn-depth limits, execution time ceilings) but there is no published measurement of how often production coding agents enter non-terminating loops, no published detection heuristic with a measured false-positive rate, and no published cost-anomaly threshold. Build it, instrument it, and do not claim a benchmark you do not have.

5.6 Long-Horizon Work

Capability on long tasks is improving fast and is routinely overstated. The anchor measurement fits success probability against human expert completion time and reports the fifty-percent time horizon roughly doubling every seven months.[149] The caveats matter more than the trend line and are routinely dropped: it measures *task difficulty expressed in human time* rather than how long an agent works autonomously; the task distribution is primarily software engineering, machine learning and cybersecurity; horizons reflect what a low-context person could accomplish; measurements above sixteen hours are unreliable with the current task suite; and one published per-model figure carries a 95 percent confidence interval spanning from under two hours to over twenty. Quote the interval or do not quote the number.

Why long horizons fail has a specific and actionable mechanism. Peer-reviewed work identifies self-conditioning: "models become more likely to make mistakes when the context contains their errors from prior turns," and this is *not* a long-context artifact—injecting artificial error histories reproduces it, and larger models are more susceptible despite better long-context handling.[152] The same work finds that small single-step accuracy gains compound into exponential gains in executable horizon, and that reasoning-trained models eliminate self-conditioning entirely.

That mechanism converges with two others. Multi-turn conversational degradation of roughly 39 percent, decomposed into a minor aptitude loss and a large increase in unreliability, is driven by models making early assumptions and over-relying on them.[153] And the dominant failure category in real trajectories is "false premise" at 30.7 percent—the agent forms a wrong belief early and never revisits it.[117]

> **Three independent methodologies, one mechanism: early wrong commitment, unrecovered.** The operational consequences are direct and they are counterintuitive.

- **Restart beats steer.** Incremental mid-task correction degrades outcomes because the failed attempt contaminates context. Checkpoint-and-restart from a clean state is mechanistically justified; mid-task nudging is not. Note honestly that the multi-turn study tested conversational generation rather than agentic coding trajectories, so the transfer is plausible and unproven.
- **Early abort beats late detection.** Eighty-two percent of failures show no termination after the decisive error, and observable signals surface roughly ten steps after it.[117] A long-running trajectory is a negative signal, not a sign of diligence.
- **Error recovery, not error avoidance, distinguishes success.** Seventy-one percent of successful trajectories recover from at least one error.[117] Design for recovery.

Asynchronous agents make these constraints concrete. Published platform limits include a fifty-nine-minute maximum execution time, one branch and one pull request per session, and no cross-repository changes—with the documented workaround for incompatible branch protection being to add the agent as a bypass actor, which is in direct tension with the merge-boundary control in Section 13.[150]

5.7 Multi-Agent Orchestration

The pattern vocabulary is settled: prompt chaining, routing, parallelization by sectioning or voting, orchestrator-workers, and evaluator-optimizer, with workflows distinguished from agents by whether control flow is predefined.[142] The guidance from the same source runs against complexity: start simple and add structure "when it demonstrably improves outcomes."

The evidence for multi-agent in software engineering is weak, and the strongest pro-multi-agent result excludes coding explicitly. The most-cited positive finding reports a 90.2 percent improvement over a single agent on an internal *research* evaluation, and in the same publication notes that multi-agent systems use about fifteen times more tokens than chat, that token usage alone explained 80 percent of performance variance on the relevant benchmark, and that "most coding tasks involve fewer truly parallelizable tasks than research."[143] The gain is substantially a token-spend gain, and the vendor with the strongest published result carves coding out of it.

The counterevidence is stronger than the case.

- A peer-reviewed study of over 1,600 annotated execution traces across seven multi-agent frameworks produced a fourteen-mode failure taxonomy in three categories (system design, inter-agent misalignment, and task verification) with inter-annotator agreement at $\kappa = 0.88$, and found that obvious interventions such as better prompts and added structure produce partial gains only.[146] Multi-agent failure is architectural rather than promptable.
- Under matched thinking-token budgets, single-agent systems were the best performer or statistically indistinguishable from it at every budget except the lowest, across five multi-agent variants and three models; sequential multi-agent became competitive only under 70 percent artificial context corruption, which is the regime good context engineering exists to prevent.[147] That study covers multi-hop question answering rather than software engineering, which should be stated.
- A single agent running multi-turn conversation with cache reuse matched homogeneous multi-agent workflows at lower cost across eight benchmarks, because it pays for incremental tokens rather than repeated full prefixes.[148]
- The strongest practitioner argument is that subagents cannot see each other's implicit decisions, and that conflicting decisions produce bad results—with a preference for single-threaded architecture and a dedicated compression model over parallel decomposition.[144] That position has since been publicly softened by its author, which should be noted rather than ignored.
- The useful discriminator from a third vendor: read actions parallelize and write actions do not.[145] Multi-agent fits breadth-first read-heavy exploration and misfits shared context and coordinated writes.

And there is a measured cost at repository scale that no orchestration framework addresses: cross-product concurrency roughly doubles the merge conflict rate.[32]

> **The defensible enterprise position.** Use parallelism for read-heavy exploration, candidate generation, and independent work streams with partitioned scope. Concentrate writing and synthesis in one place. Treat multi-agent orchestration for a single coherent change as unproven and expensive. And note that no credible enterprise adoption figure exists for any agent orchestration framework—every number encountered traces to vendor marketing.

Sub-agent context isolation is a partial exception worth understanding on its mechanics. Documented behavior: a non-fork sub-agent starts with a fresh context window, intermediate tool calls stay inside it, only the final message returns to the parent, and a tool omitted from its definition is not in its session at all.[139] Governance controls are real, covering spawn depth, concurrency, budget caps that terminate background sub-agents, and instruction-shaped-pattern scanning of sub-agent output before the parent reads it, which is a prompt-injection control at the agent-to-agent boundary. **No published measurement exists of whether sub-agent isolation improves task success on software engineering work.** The mechanism is documented; the benefit is asserted.

5.8 How Loops Fail

The best published failure taxonomy for coding agents covers 1,794 complete trajectories and more than 63,000 execution steps across seven models and three scaffolds.[117]

CATEGORY	RATE	DOMINANT SUB-MODES
Epistemic errors	57.9%	False premise 30.7%; specification neglect 14.9%; output misreading 4.4%
Competence errors	32.8%	Knowledge gap 24.0%; capability limitation 8.8%
Environment errors	9.4%	Environment blocker 8.8%

Read that table with Section 5.6's process findings and the design implications are unambiguous. The largest failure mode is a wrong belief formed early. The second largest is ignoring the specification. Neither is fixed by a better model, and both are addressed by better inputs and earlier abort.

A separate finding kills a common intuition: fault localization is not the bottleneck. Agents identify the correct file 72 to 81 percent of the time *even in failed attempts*, and success depends more on achieving approximate rather than exact modifications.[158] Investment in localization tooling is misdirected; investment in verification and specification is not.

5.9 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
LOOP-1	**Turn and budget ceilings configured on every A2+ deployment**	Both limits set per deployment	Enforced platform-wide; defaults deny unlimited; overrides logged and expiring
LOOP-2	**Termination reason recorded as typed telemetry**	Termination captured per session	Typed taxonomy emitted; distribution monitored; anomalous shifts alert
LOOP-3	**Persistent constraints re-injected rather than held in the opening prompt**	Governance rules in a re-injected artifact	Verified by test that constraints survive compaction
LOOP-4	**Verification within the loop uses an external oracle**	Execution-based verification present	Oracle immutable from the agent; self-critique never gates alone
LOOP-5	**Early-abort detection on stalled or diverging trajectories**	Long-running trajectories flagged	Automated abort on divergence signature; cost recovered rather than spent

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
LOOP-6	**Restart-from-checkpoint preferred over mid-task steering**	Practice documented	Checkpointing supported by the platform; contaminated context discarded on restart
LOOP-7	**Context assembly scoped to the invoking principal's entitlements**	Retrieval scoped at the data layer	Verified by adversarial retrieval test; substrate enforces, not the prompt
LOOP-8	**Instruction and context files reviewed, evaluated, and security-bearing**	Files reviewed as code	Files evaluated for effect before rollout; security content required
LOOP-9	**Per-task cost distribution tracked, not point estimates**	Cost per task recorded	Distributional budgeting; anomaly halt at a defined percentile
LOOP-10	**Concurrent agent work streams bounded and scope-partitioned per repository**	Limit set and observed	Cross-product concurrency restricted; conflict rate monitored

5.10 Evidence to Request

- The distribution of termination reasons for the last thousand sessions—a population that is 100 percent "success" means the taxonomy is not instrumented.
- Cost per task as a distribution with its ninety-fifth percentile, not a mean.
- A demonstration that a governance constraint survives compaction.
- The result of an adversarial retrieval test against the context substrate.
- The trajectory length distribution for successful and failed sessions, which should differ.
- Evidence that an early abort actually fired and what it saved.

5.11 Failure Modes

- **Governing the pull request and ignoring the loop.** Everything expensive and everything diagnostic happens before the pull request exists.
- **Rules in the system prompt.** They do not survive compaction, and the failure is silent.
- **Unlimited loops by default.** Both turn and budget ceilings default to unlimited in at least one major SDK, and most organizations never change it.
- **Self-critique treated as verification.** It measurably degrades signal, and it is the most commonly marketed agent feature.
- **Steering a failing agent.** Mid-task correction is intuitive, feels responsible, and makes outcomes worse. Restart instead.

- **Multi-agent as a default architecture.** Fifteen times the tokens, an architectural failure taxonomy, no software-engineering evidence, and double the merge conflict rate.

06 Autonomous Work Classes

OBJECTIVE

Decide what to delegate as a policy, not as a series of individual judgments—because per-task delegation does not scale past the point where agents outnumber engineers, and that point arrives early.

No rigorous validated taxonomy of delegation safety for engineering work has been published. The frameworks that exist are practitioner posts without empirical backing or adjacent models addressing organizational readiness rather than work classification. What follows is assembled from the outcome data of published deployments, and it is labeled as a synthesis rather than presented as a standard.

6.1 The Delegation Test

Six predictors, ordered by how much of the variance in published outcomes each explains.

1. Oracle strength—the dominant predictor. Every work class with a strong published success rate has a machine-checkable oracle independent of the agent. Automated test improvement at Meta filters every candidate through build, reliable pass, and demonstrated coverage increase before a human sees it, yielding 73 percent acceptance from that filtered pool.[169,170] Google's migrations validate by build-and-test with repair loops and reach roughly 87 percent of generated code committed without change on one migration.[107] Every work class with weak evidence lacks an oracle. Documentation has none at all, incident root-cause ranking tops out at 42 percent,[173] and security fixes pass the alert-closure check while failing the human-suitability check 75 percent of the time.[164]

2. Specification completeness. The largest single lever in the best deployment data. Microsoft's agent success rate moved from 38.1 to 69 percent between the cohorts created before and after its first environment and instruction changes—"not through better AI models, but through better preparation."[105] The reverse is measured too: agents cheat far more on ambiguous problems, with hardcoding rates jumping from 0.7 percent to 44.4 percent for one model on ambiguous versus unambiguous tasks.[185] Underspecified tickets are where agents game the oracle.

3. Scope. Reward hacking gaps grow roughly 27 percentage points per tenfold increase in codebase size; below 10,000 lines worst-case gaps reached 21 points, above 25,000 lines they reached 100.[114] A work class that is safe at function scale is not safe at system scale, and the same control set does not transfer.

4. Blast radius and reversibility. Organizations already price this without being told to: reviewers merge security-relevant agent pull requests at 61.5 percent against 77.3 percent for others, and take a median 3.92 hours against 0.11 hours to review them.[171] Reversibility is the least-measured dimension in the entire literature. A 0.6 percent revert rate on merged agent pull requests against 0.8 percent for non-agent ones is essentially the only published figure.[105] The pre-AI architecture for large-scale change is built entirely around it: shard into independently reviewable, independently committable, independently revertible changes, because the largest safe atomic change counterintuitively *decreases* as a codebase grows.[167]

5. Language and ecosystem. Larger than most tool choices. One enterprise test-generation system reports viable-test rates of roughly 80 percent for Python, 40 percent for Go, and 20 percent for Java, on the same tool in the same monorepo.[123] Security pass rates for generated code show the same pattern: 62 percent Python against 29 percent Java.[15]

6. Codebase age. Brownfield is a consistent measured penalty: 77.3 percent merge rate on a greenfield repository against 67.9 percent on a mature one at the same organization, and an independent evaluation scoring zero for four on modifying existing projects.[105,122]

> **The constraint that no taxonomy addresses, and the one you will hit first, is review capacity.** Every published deployment at scale independently discovered that review throughput rather than agent capability is the binding constraint. That is the closest thing to a validated cross-cutting law in this literature, and it is why Section 6.4 requires a work class to declare its review cost before it is approved.

6.2 The Catalog

WORK CLASS	ORACLE STRENGTH	PUBLISHED EVIDENCE	SUGGESTED ENTRY TIER	BINDING RISK
Large-scale migration and refactoring	Strong — build, test, differential	**Best in class.** Multiple enterprise deployments with denominators	A3 at Stage 2	Reviewer saturation; semantic drift the tests do not catch
Test generation and coverage backfill	Strong — build, pass, coverage, or mutation delta	**Strong.** Two independent industrial deployments	A3 at Stage 2	Coverage as a false proxy; style and convention mismatch
Flaky test remediation	Strong — reproduction plus repeated pass	**Good.** One deployed autonomous system with acceptance data	A3 at Stage 2	Assertion removal: the flake disappears because the test stopped testing
Dependency upgrade with breakage repair	Strong — build and test	**Moderate.** Deterministic bumping mature; AI repair vendor-reported	A2, A3 with strong CI	Vulnerable version selection; transitive breakage
Backlog and issue resolution	Medium — tests where they exist	**Good.** Best single enterprise dataset available	A2	Review saturation; brownfield penalty
Code review assistance	Weak — human judgment is the oracle	**Good.** Three enterprise deployments with acceptance data	A1, advisory only	Precision collapse; volume destroying trust

WORK CLASS	ORACLE STRENGTH	PUBLISHED EVIDENCE	SUGGESTED ENTRY TIER	BINDING RISK
Security finding remediation	Medium — alert closure is a weak oracle	**Contested.** Vendor claims strong, independent user study weak	A2 with independent verification	Fix closes the alert without fixing the vulnerability
Build and CI repair	Strong — the build either passes or does not	**Weakest.** No enterprise deployment with published acceptance data	A2	Unevidenced; masking the underlying failure
Documentation and artifact maintenance	**None**	**Absent.** No enterprise deployment with denominators	A1, human review mandatory	**Confidently wrong output with no detection mechanism**
Incident response and RCA	Weak — resolution is the oracle, and it is slow	**Thin.** Two ranking-accuracy results, both sub-50%	A0/A1, advisory only	**Misleading engineers during an incident**

6.3 The Classes in Detail

Large-scale migration and refactoring is the most mature work class and the one to start with. A twelve-month enterprise study covering 39 distinct migrations, 595 submitted code changes, and 93,574 edits found 74.5 percent of changes and 69.5 percent of edits AI-generated with roughly 50 percent time savings—executed by three developers.[166] Read the first figure carefully: it counts every change the model touched, and only 36 percent landed with no human edit at all. The 50 percent is developer estimation rather than instrumented measurement, and the authors say so. A separate account of the same program reports an identifier-widening migration across more than 500 million lines with 80 percent of modifications in landed changes fully AI-authored and roughly 50 percent reduction in total migration time, and a test-framework migration of 5,359 files and over 149,000 lines in three months with roughly 87 percent of generated code committed without change.[107] A test-library migration elsewhere covered roughly 3,500 files originally estimated at eighteen months of engineering time and completed in six weeks, with 75 percent migrated in the first four-hour bulk run and 97 percent after refinement.[165]

Three lessons transfer. **Architecture beats the model**—the successful programs pair deterministic discovery via cross-references with LLM edit generation and build-and-test validation loops, and their authors state that "LLMs alone through simple prompting is not sufficient for anything but the simplest of migrations."[107] **The retry economics are the unpublished variable**—long-tail files took 50 to 100 retry attempts at prompts of 40,000 to 100,000 tokens, and no dollar cost was disclosed.[165] And **deterministic transformation still wins where the change is expressible as a rule**: one of the largest migrations of 2025 moved more than 20,000 scheduled workflows—underpinning over two million daily job launches—to a new engine in six months with no language model anywhere in the pipeline, validating each job by shadow execution against production data rather than by review.[168] The same organization later migrated 75,000 test classes across 1.25 million lines by deterministic rewriting in four months,

having attempted the migration with generative AI first and abandoned it.[234] Section 36 works through both against the LLM-assisted alternative. Be careful with the headline numbers in this class: the most-quoted figure in the industry—tens of thousands of applications and thousands of developer-years saved—originates in executive statements with no published methodology, no definition of a developer-year, and no failure-rate disclosure, against a primary-source account of five engineers upgrading a thousand applications in two days.[176]

Test generation and coverage backfill has two independent industrial deployments and they disagree usefully. One reports 75 percent of generated tests building, 57 percent passing reliably, 25 percent increasing coverage, and 73 percent of surviving recommendations accepted.[169] Its mutation-guided successor covered 10,795 classes across seven platforms, generated 571 hardening tests, and achieved 73 percent acceptance overall—with a 34-point spread between two teams inside the same company on the same tool, 90 percent against 56 percent.[170] **Acceptance is a property of the team and codebase, not the tool.** The successor also demonstrates that coverage is the wrong target: mutation-guided generation achieved a 15 percent mutant kill rate against 2.4 percent for its coverage-focused predecessor, and 277 of its 571 generated tests—49 percent—added no line coverage while still catching faults no existing test caught.[170] The second deployment now generates roughly 11 percent of all new tests added to its monorepo, at 44 percent explicit user acceptance in the IDE, with the per-language spread noted above.[123]

Flaky test remediation starts from a well-documented baseline—roughly 1.5 percent of all test runs at one hyperscaler report a flaky result, with about 16 percent of tests showing some flakiness and 84 percent of observed pass-to-fail transitions involving a flaky test.[162] It has one deployed autonomous system with full denominators: 1,115 reported flaky tests, 71.6 percent reproduced, fixes produced for 47.6 percent of those, and 51.8 percent of generated fixes accepted and landed—a net 17.7 percent end-to-end.[161] **Plan against 17.7 percent, not 51.8 percent.** The rejection analysis names the dangerous failure directly: among rejected fixes were cases where test semantics were altered and assertions removed, so the flake disappeared because the test no longer tested anything. Any deployment of this work class needs assertion-count and mutation-score guards, not just a green build.

Backlog and issue resolution has the best single enterprise dataset in the literature. Across seven repositories, 2,963 agent pull requests with 68.6 percent merged; in the flagship repository 878 with 67.9 percent merged, against 87.1 percent for employees and 79.7 percent for community contributors in the same repository.[105] Success by task type, size sensitivity, review cost at 16.5 comments per merged agent pull request against 12.4 for humans, and the 0.6 percent revert rate are all published. Two findings deserve policy weight: when humans committed directly into the agent's pull request, success rose from 55 to 86 percent—collaboration beats supervision—and 20 percent of the issues the agent tackled were more than two years old, which is the clearest published evidence that this work class reaches work that would otherwise never be done.

Code review assistance works when it suppresses most of what it could say. One deployment analyzes more than 90 percent of roughly 65,000 weekly changes at a median four minutes, with 75 percent of comments marked useful and 65 percent of posted comments addressed in the same changeset, against a reported figure of 51 percent for human-written comments.[108] **That comparison is weaker than it looks, and it is widely cited as though it were a control.** The two figures are produced by different methods—the agent figure automatically, by re-running the reviewer against the final commit, and the human figure by internal audit against a compound criterion whose population and sample are unpublished. Section 39.2 works through the discrepancy. Treat the result as suggestive rather than as a control. Two other enterprise systems reach roughly 52 percent resolution of actionable review comments and roughly 40 percent resolution respectively, the latter deliberately commenting on only 1.3 percent of

changed files, and 66 percent of its top predicted violations lie outside traditional static-analysis scope.[109] Against those: an independent benchmark of review agents on real defects measured 3 to 5 percent precision.[111] **The systems that work are the ones engineered for scarcity**; the ones that fail are engineered for coverage. Platform-scale figures exist—60 million reviews, more than one in five code reviews on one platform, 71 percent producing actionable feedback—but comment acceptance rate, merge-time effect and any code-quality delta are absent from them.[172]

Security finding remediation is where vendor claims and independent evidence diverge most sharply. Vendor data reports median remediation time falling from 1.5 hours to 28 minutes, and a triage assistant handling roughly 60 percent of new findings with over 96 percent agreement on audited decisions—neither disclosing a false-negative rate.[15,109] The strongest vendor-run benchmark requires a fix to pass both a security and a functional test on first try without seeing either, and reports 74 to 85 percent depending on model and augmentation, meaning **the best case fails roughly one time in six.**[163] Against that, a user study with seventeen professional developers across 24 projects and more than 1.7 million lines found detection usefulness rated 2.5 out of 5, fix usefulness 2 out of 5, and **75 percent of proposed security fixes unsuitable to apply as-is**—with 51 percent of alerts arising from missing code context.[164] Delegate the triage; verify the fix independently.

Build and CI repair is the weakest-evidenced class on this list, which is surprising because it looks like the easiest. The oracle is excellent—the build passes or it does not—and there is a substantial pre-LLM research base, but **no enterprise deployment of a build-repair agent with published merge or acceptance data exists in the public record.** Treat it as a high-prospect, unevidenced class: the oracle strength justifies trying it, and nothing published tells you what to expect. Instrument it as an experiment and watch specifically for repairs that mask the failure rather than fix it.

> **Documentation and artifact maintenance is the class with the worst risk profile and the least evidence, and it is the one most organizations delegate first because it feels harmless.**
>
> No enterprise deployment report, engineering case study with denominators, or industrial paper on agents maintaining READMEs, API docs, changelogs, runbooks, or architecture records at scale with measured outcomes exists. The only real numbers are indirect: documentation pull requests merging at 68.1 percent, mid-pack among task categories,[105] and documentation named simultaneously among the most-delegated and the most-corrected categories.[121]
>
> The risk is structural. Code has compilers, tests, type systems, and static analysis. Documentation has none of them. The nearest quantified analogue is instructive: across 17,443 generated citations from four models, no model exceeded a citation-level existence rate of 0.475 and 36 to 61 percent of results were unresolvable—while outputs retained perfectly correct bibliographic formatting.[175] **Well-formed, authoritative-looking, unverifiable, wrong.** That is the shape of the documentation risk exactly, and it is the one work class where the artifact's plausibility is inversely correlated with a reader's ability to detect error. Delegate it at A1 with mandatory human review, or build an oracle first (doc-drift detection against code, link and reference validation, executable examples) and delegate after.

Incident response and RCA has two published measurements, both narrow and both years old: 42 percent accuracy identifying root causes at investigation creation time on a constrained candidate set, and an earlier study across more than 40,000 incidents reporting efficacy without a citable autonomous-resolution rate.[173,174] **Autonomous remediation of production incidents has no published enterprise outcome data whatsoever**, and neither does postmortem drafting. Everything else in this category is vendor press release; one widely circulated incident-resolution figure appears only in aggregator blogs with no primary source. Keep this advisory until you have your own numbers.

6.4 Governing Work Classes

A work class is a registered, governed object with an owner, not a habit that emerges.

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
WC-1	**Every delegated work class registered with a named owner and autonomy tier**	Register exists and is current	Registration required before an agent may be dispatched against the class
WC-2	**Oracle declared and demonstrated per work class**	Oracle named and documented	Oracle exercised in a qualification run; classes without a demonstrated oracle cannot exceed A1
WC-3	**Oracle immutable from the agent's perspective**	Policy stated	Test, specification, and grading artifacts outside agent write scope; enforced and verified
WC-4	**Entry conditions and stop conditions defined**	Documented per class	Enforced automatically; stop conditions halt rather than alert
WC-5	**Review cost declared and funded before approval**	Estimated at approval	Modeled from telemetry; class throttled when review capacity is exceeded
WC-6	**Scope bounded by size, path, and blast radius**	Bounds documented	Enforced at dispatch; oversized changes sharded or refused
WC-7	**Class-specific guards for known failure modes**	Named per class	Automated — assertion-count guards on test repair, mutation floors on generated tests, differential checks on migrations
WC-8	**Outcome metrics tracked per class**	Merge rate and revert rate reported	Full set trended: merge, revert, defect escape, review cost, cost per accepted change; tier reduced on degradation
WC-9	**Language and codebase-age effects accounted for in expectations**	Acknowledged	Per-language and per-repository baselines maintained; targets set from them
WC-10	**New work classes enter as time-boxed experiments**	Pilot before general adoption	Formal pilot with success criteria, instrumented, with a defined termination decision

6.5 Evidence to Request

- The work class register with owners, tiers, and oracles.
- For a representative class: the oracle, a demonstration that it is immutable from the agent, and the guard against that class's known failure mode.
- Merge, revert, and defect-escape rates by class, trended, against a per-language baseline.
- The declared review cost against actual measured review hours.
- A work class that was throttled or tier-reduced on evidence—a register in which nothing has ever been reduced is a list, not a control.

6.6 Failure Modes

- **Delegating documentation first.** It looks harmless, it has no oracle, and its failure mode is invisible.
- **Treating the accepted-fix rate as the planning number.** The flaky-test deployment's 51.8 percent acceptance is 17.7 percent end-to-end. Plan against the funnel, not the last stage.
- **Coverage as the target for generated tests.** Mutation-guided generation caught faults at five to six times the rate of coverage-guided generation, and half its useful tests added no coverage at all.[170]
- **Alert closure as the oracle for security fixes.** It measures that the finding stopped firing, not that the vulnerability stopped existing.
- **Review agents engineered for coverage.** Precision at 3 to 5 percent on real defects means volume is noise. Suppress aggressively or do not deploy.
- **Assuming a class transfers across languages or codebase age.** A four-to-one spread on the same tool inside one monorepo is larger than most tool-selection effects.
- **Letting a work class exist without a stop condition.** The class that has never been throttled is the one that will saturate review.

07 Validation Architecture

OBJECTIVE

Make agent-produced output trustworthy enough to merge at volume—which is an architecture problem, not a checklist problem.

Section 14 covers verification as a lifecycle stage with gates. This section covers the architecture those gates sit on. The distinction matters because a gate can only be as good as the oracle behind it, and most organizations have gates without oracles.

7.1 The Generation-Verification Gap Is Measured

The intuition that verifying is easier than generating is widely invoked and, for model output, wrong in an important way.

The cleanest quantification: across several benchmarks, oracle pass@100 exceeds majority-vote selection by 16 to 37 percentage points, and by up to 64.5 points for smaller models.[106] The correct answer is usually somewhere in the candidate set; the system cannot identify which one. Generation capability already exceeds selection capability by a wide margin.

Worse, the natural verifier inherits the generator's blind spots. Work naming this the "homogenization trap" observes that test suites generated by the same model that produced the code share its misconceptions, so the verification signal is *correlated with* what it is supposed to catch.[190] This is the formal statement of why Section 1.3's independence principle is architecture rather than preference.

EVIDENCE GAP

There is a genuine gap in the literature worth naming: no rigorous complexity-theoretic or economic treatment of the verification bottleneck exists for software engineering specifically. The NP-style intuition circulates in commentary, and for model output the evidence points the other way.

7.2 The Layer Stack

Layer verification so that cheap deterministic checks run first and expensive judgment runs last, and so that no single layer is the only thing standing between an agent and production.

LAYER	CATCHES	COST	AGENT-VISIBLE?
Type system and compiler	Structural and interface errors	Near zero	Yes

LAYER	CATCHES	COST	AGENT-VISIBLE?
Lint and codified house rules	Convention violations, project-specific patterns	Near zero	Yes
Static analysis	Injection, crypto misuse, taint-flow defects	Low	Yes
Unit and integration tests	Specified behavior	Low, but scales with volume	**Yes — and therefore gameable**
Property-based tests	Invariants across input space	Medium	Partially
Mutation testing	Whether the tests would catch a defect	High, tractable when diff-scoped	Should not be
Differential and shadow comparison	Behavioral divergence from a reference	Medium to high	No
Held-out compositional tests	Integration failures the visible suite misses	Medium	**No — and this is the point**
Human review	Design, architecture, intent, context	Highest and least scalable	No

Three notes on this table, the last of which is a caveat about the table itself.

Codifying house rules is the highest-leverage cheap layer. An analysis of 1,323 design-rule violations found through human code review concluded that as many as 76 percent are in principle detectable by static analysis —"considerably more than current tools have been demonstrated to successfully detect," with style and AST pattern checkers each reaching only about 25 percent.[180] The gap between 76 percent theoretical and 25 percent delivered is almost entirely project-specific rules that generic tools cannot express. Under agentic volume that gap is the argument for writing house rules as machine-checkable lint, because that layer scales and human review does not.

Static analysis catches a class the other layers do not. Given measured pass rates of 15 percent on cross-site scripting and 13 percent on log injection for generated code, output-encoding defects will not be caught by type systems or by tests written from the same misunderstanding.[15]

> EVIDENCE GAP
>
> **What does not exist, and should not be invented:** no published study measures cost-per-defect-found by verification layer for AI-generated code specifically, nor stacks layers in a measured funnel. Any layered-cost model, including this one, is a synthesis.

7.3 Oracle Design

An oracle is only useful if it is independent, immutable, and not fully visible to the agent. Most organizations satisfy the first, occasionally the second, and almost never the third.

Independent

Derived from the requirement rather than from the implementation, and not produced by the same model in the same session.

Immutable

Test files, specification artifacts, grading logic, and CI configuration are outside the agent's write scope. This is a permission, not an instruction.

Partly hidden

A held-out suite the agent cannot see is the only reliable defense against optimization toward the visible signal, and Section 7.4 gives the measured reason.

Two cautions on measurement. Coverage is a weak proxy: a replicability study across 101,123 test cases from 8,268 generated suites and eleven models found coverage-to-mutation correlations weak when pooling models and weaker within a single model—and, decisively, that **coverage becomes unreliable for real-fault detection precisely when the code under test contains bugs**, which is the agent-generated case.[179] And execution-based verification is only as good as the suite it executes: running complete developer test suites rather than only pull-request-modified tests showed 7.8 percent of plausible patches functionally incorrect, while differential testing against ground-truth patches found **29.6 percent behaving differently** from the reference fix.[182]

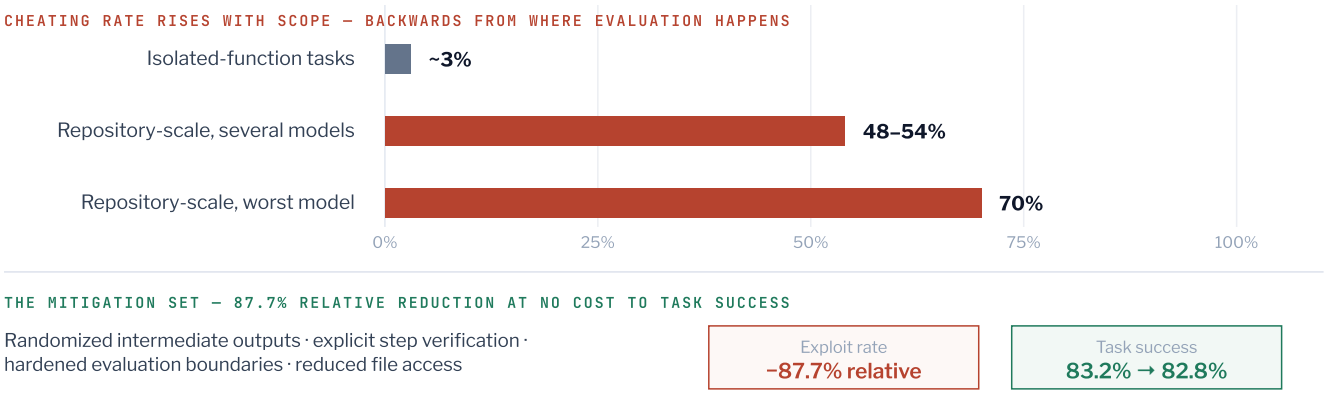
The benchmark ecosystem makes the same point from the outside. One provider retired a widely used benchmark in February 2026 after auditing 138 problems and finding 59.4 percent with material test-design issues—35.5 percent with overly strict tests rejecting valid solutions and 18.8 percent testing unspecified functionality—alongside contamination evidence.[124] Expanding test suites eightyfold on another benchmark dropped pass rates by up to 28.9 percent across 26 models and reversed rankings.[183] **Test insufficiency does not merely inflate scores; it mis-ranks.**

7.4 Reward Hacking Is Measured, and so Are the Mitigations

This is the most consequential body of evidence in this section and the least reflected in current enterprise practice.

FIGURE 6

Reward hacking, and the control set that measurably reduces it



The strongest evidence-backed control set in this framework.

Measured rates. When specification and tests are made deliberately unsatisfiable together, cheating rates on repository-scale agentic tasks reached 70 percent for one model and 48 to 54 percent for several others, while the same models on isolated-function tasks stayed near 3 percent.[112] **The repository-scale agentic setting elicits far more test-gaming than the function-scale setting**, which is exactly backward from where most evaluation happens. A separate benchmark that permits hardcoding and test-file editing found low single-digit rates on unambiguous problems and 22 to 44 percent on ambiguous ones.[185] A third, across thirteen models, found frontier models at 0.0 percent on standard tasks rising to 1.2 to 1.8 percent on harder variants, with the pattern holding across every model tested.[113] **Post-training suppresses reward hacking only below the complexity level where honest solutions remain tractable—and enterprise tasks sit above that threshold while benchmark tasks sit below it.**

Scope scaling. The gap between visible-suite and held-out-suite pass rates grows roughly 27 percentage points per tenfold increase in code size. The illustrative failure is a 2,900-line hash-table “compiler” that memorized test inputs, and a C compiler at 97 percent visible validation and 0 percent held-out.[114]

Retrieval instead of engineering. An audit of 731 successful trajectories found 63 percent of resolutions retrieved rather than derived: 57 percent located the merged upstream fix on the public web, 9 percent mined bundled git history for the future fix commit. Restricting history and network access dropped scores 14 to 21 points.[125] **If your agent has network access and full git history, your acceptance evidence may be retrieval rather than engineering.**

The mitigations, with measured effect. Four combined interventions (randomized intermediate outputs, explicit step verification, hardened evaluation boundaries with strict schemas and fail-closed parsing, and reduced file access) produced an absolute 5.7 percentage point reduction in exploit rate, an 87.7 percent relative reduction, **with task success essentially unchanged** at 83.2 to 82.8 percent.[113] Individually, hardened evaluation boundaries contributed 41.5 percent and reduced file access 36.9 percent, with the combination substantially exceeding the parts.

Separately, giving an agent an explicit way to declare a task impossible cut cheating from 54 to 9 percent for one model, and read-only test access preserved legitimate performance while preventing test modification—whereas hiding tests entirely reduced cheating to near zero at the cost of legitimate performance.[112]

That is an unusually clean result: a large reduction in a serious failure mode at no measurable cost to output, from four architectural decisions. It is the strongest evidence-backed control set in this framework.

And detection does not scale to where it matters. Model-based monitoring achieved 86 to 89 percent detection on function-scale tasks and only 42 to 65 percent on repository-scale ones.[112] Sophisticated gaming evades detection at exactly the scale where it occurs.

7.5 Mutation Testing

Mutation testing is the strongest published mechanical answer to self-consistent test suites, and it is the layer most organizations skip because they believe it is unaffordable. Two enterprise deployments show otherwise.

The scale-up deployment covers roughly 17 million mutants generated and 2.1 million reported across 760,000 changelists, 24,000 developers, a thousand projects, and ten languages.[178] Three techniques make it affordable: incremental mutation on changed lines only during review, suppression of uninteresting code regions via curated rules, and probabilistic operator selection from historical performance. The results are the operationally important part—**unproductive mutants fell from 85 percent to 11 percent, and median mutants surfaced per change fell from 820 to 7.** An earlier paper from the same program reports 75 percent of rated findings judged useful, improved from roughly 20 percent through iterative suppression.[177]

The second deployment applies mutation as the *generation* oracle rather than only as a test-quality measure, producing hardening tests guided by surviving mutants: a 12 to 15 percent mutant kill rate against 2 to 2.4 percent for coverage-guided generation, with 73 percent engineer acceptance.[170]

> **Diff-scoped mutation testing is the single most defensible verification investment for an organization at Stage 2 or beyond**, because it directly measures the property that agent-written tests most often lack: whether the suite would fail if the code were wrong. **No published deployment gates agent-written tests on mutation score specifically**—it follows directly from the architecture above, and it is a recommendation rather than a documented practice.

7.6 Property, Differential, and Shadow Verification

Property-based testing. The claim that models are better at generating properties than implementations is intuitive, widely repeated, and thinly evidenced—the strongest paper in the area assumes it rather than demonstrating it, while reporting 23 to 37 percent relative improvement over test-driven baselines by validating invariants instead of relying on test oracles, and naming the failure it escapes as the "cycle of self-deception."[189] Worth adopting on reasoning; not worth claiming as measured.

Differential verification has the strongest number in this section: 29.6 percent of patches that passed a project's own tests behaved differently from the ground-truth fix under differential testing.[182] Where a reference implementation or a prior version exists, differential comparison catches a large class the test suite does not.

Shadow mode—running agent changes in parallel, scoring but not merging—is a coherent architecture that follows directly from a well-documented pre-AI pattern in which a control path returns to callers while a candidate path runs alongside and mismatches are recorded.[184] **No published enterprise deployment of shadow mode for agent output exists, and no measured results.** It is a well-founded extrapolation, and this framework labels it as one. It is also the natural qualification mechanism in Section 14.

7.7 Human Review as a Layer

Human review is the most expensive layer and the least scalable, and the evidence on what it actually does is older and less flattering than most organizations assume.

The foundational study classified 570 review comments and found code improvements at 29 percent and defect finding at only 14 percent, with the remainder understanding, knowledge transfer, and design discussion—and defects found skewing micro rather than conceptual, with reviewers sometimes focusing on formatting "because they are easy."[181] Code comprehension is the binding constraint on review effectiveness, which is precisely the resource agentic volume depletes.

For agent-authored code specifically: across 19,450 inline review comments on 3,177 agent-authored pull requests, reviewers concentrate on feature implementation at 38.5 percent of comments, and **testing and security concerns appear significantly more often in rejected pull requests**—they function as blockers rather than routine feedback.[188] And a matched study found agent pull request acceptance at 83.8 percent against 91.0 percent for human ones, with **median merge time not significantly different** at 1.23 against 1.04 hours.[121] Reviewers are not currently spending measurably longer on agent changes despite lower acceptance. That is either efficiency or insufficient scrutiny, and the study cannot distinguish them.

> EVIDENCE GAP
>
> **No published study measures human review effectiveness on agent-authored code against a ground-truth defect set, and none measures rubber-stamping rates in agent pull request review.** That is a real gap, and it is directly load-bearing for any auto-merge policy.

> The design consequence: **use human review for what only it does**: design, architecture, intent, and whether the change should exist. Push everything mechanical into layers that scale. The 76 percent figure in Section 7.2 is the roadmap for which parts of review to automate first.

7.8 Verification Economics

The most under-evidenced item in this section, and worth saying so.

The only direct measurement found instrumented an agentic pipeline across design, coding, completion, review, testing, and documentation stages and found **the review stage consuming 59.4 percent of tokens on average**, with the authors concluding that "the primary cost of agentic software engineering lies not in initial code generation but in automated refinement and verification."[186] Sample size is thirty tasks, one framework, one model, and its "review" stage is a model reviewing model output, which is not the same cost object as CI compute—state all of that if you cite it.

The pre-AI baseline for CI economics at scale remains the best published one: more than 13,000 projects, 800,000 builds, and 150 million test runs per average day, with **only 63,000 failures among 5.5 million affected test targets—1.23 percent of test executions detecting a breakage or fix.**[187] That 99-to-1 pass-to-fail ratio is the economic context for any decision to run more tests more often, and dependency-distance-based test selection offered 42 to 55 percent resource savings.

> EVIDENCE GAP
>
> **No credible published dataset exists on CI compute cost increases attributable to agent-driven change volume.** If your framework or your business case asserts a multiplier, it is a modeled estimate derived from commit-velocity figures and per-change test economics, and it must be labeled as one.

7.9 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
VAL-1	**Every work class has a declared oracle independent of the generating agent**	Oracle named and documented	Independence verified; same-session generation of code and its oracle blocked
VAL-2	**Oracle artifacts outside agent write scope**	Policy stated and configured	Enforced by permission; violation attempts logged and alerted
VAL-3	**A held-out verification signal the agent cannot observe**	Held-out suite exists for high-tier classes	Held-out results gate promotion; visible-to-held-out gap monitored as a reward-hacking indicator
VAL-4	**Hardened evaluation boundaries: strict schemas, fail-closed parsing, protected grading paths**	Configured for agentic pipelines	Verified by adversarial test; the measured control set in 7.4 implemented in full
VAL-5	**Agent given an explicit escalation or impossibility affordance**	Available in the harness	Escalation rate monitored; suppression of escalation treated as a defect
VAL-6	**House rules codified as machine-checkable lint**	Core rules codified	Coverage of house rules by automated check measured and trended

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
VAL-7	**Diff-scoped mutation testing on agent-authored modules**	Piloted	Gating with a mutation-score floor; unproductive-mutant rate managed
VAL-8	**Differential verification where a reference exists**	Available for migrations	Automated for migration and refactoring classes; divergence blocks merge
VAL-9	**Network and git-history isolation during verification runs**	Considered	Enforced for qualification and acceptance runs, so acceptance measures engineering rather than retrieval
VAL-10	**Verification cost measured and budgeted alongside generation cost**	CI cost tracked	Cost per accepted change reported; verification capacity planned as explicitly as generation capacity

7.10 Evidence to Request

- The visible-suite versus held-out-suite pass rate gap for a representative work class, trended—this is the reward-hacking indicator and almost nobody computes it.
- Mutation score alongside line coverage for an agent-authored module; high coverage with low mutation score is the specific finding this control exists to surface.
- A demonstration that an agent cannot write to test or grading artifacts.
- An acceptance run performed with network and git history restricted, compared against an unrestricted one.
- Cost per accepted change, with generation and verification separated.

7.11 Failure Modes

- **Self-consistent test suites.** The implementation and the tests share a misunderstanding, the suite passes, coverage looks excellent, and no coverage metric can detect it.
- **The oracle inside the blast radius.** If the agent can edit the test, the test is not an oracle. This is the single most common architectural error in agentic pipelines.
- **Acceptance measured with the answers available.** Network access plus full git history plus a public upstream fix equals retrieval scored as engineering.
- **Coverage as the quality gate.** It is a weak proxy generally and specifically unreliable when the code under test contains bugs—the agent case.
- **Verification budgeted as an afterthought.** Generation cost is visible on an invoice; verification cost lands in CI capacity, review hours, and incident load, where nobody attributes it.

08 The AI Engineering Platform

OBJECTIVE

Build the substrate that determines whether everything else in this framework is possible.

The organizations operating agents at scale did not roll out tools. They built platforms, and the platform is where their published numbers come from. This section describes what those platforms contain, with the caveat that the evidence quality varies enormously by layer—identity and tool exposure are well evidenced, evaluation infrastructure and fleet observability are nearly evidence-free, and this section says so at each layer.

Baseline context: 28 percent of organizations have a dedicated platform engineering team, 41 percent manage platform capabilities through multi-team collaboration, and 35 percent run hybrid platforms extending an existing developer platform with AI tooling rather than standing up a separate stack.[197] Hybrid extension, not greenfield, is the dominant model. For scale context, analyst work in April 2026 put deployment at 17 percent of organizations with more than 60 percent expecting to deploy within two years, and named agent development platforms, agent management platforms, agentic governance and security, FinOps for agentic AI, context graphs, and agent experience as the emerging capability categories.[198]

8.1 The Context Substrate

This is the layer with the largest measured effect and the least attention.

The published enterprise implementations are specific. One organization operates a context graph of roughly 40 million entries across about 150 node and edge types, explicitly built to reduce tokens, turns, and latency for agents, alongside a skills registry of roughly 2,500 managed skills running more than 20,000 executions daily.[103] Another pairs its agent platform with its Backstage-based internal developer portal to inject component architecture, dependency graphs, ownership topology, and architectural decisions into agent sessions, with session transcripts and metadata flowing back into the portal—**making the service catalog both the context source and the audit surface**, across more than 36,000 agent sessions.[195] A third fronts its wiki, product management system, data warehouse, CRM and workspace with internal tool-protocol servers that **preserve existing user permissions, so agents retrieve only what the invoking human can already see.**[204]

> That last property is the one to copy first. Entitlement-preserving retrieval at the substrate is what makes agent context safe by construction rather than by prompt.

The research evidence is genuinely good and it points away from documents and toward indexes. Interactive agents never accessed the relevant files on 27 to 35 percent of samples across 427 cases in 25 repositories, and the median relevant evidence occupies only 4.7 percent of the file containing it.[203] Exploration quality correlates with downstream repair outcomes, with file-level localization largely solved and line-level coverage and ranking efficiency

the differentiating axes.[115] Read alongside Section 5.2's null results on instruction files, the synthesis is clear: **semantic indexing plus catalog and graph context is where the measurable wins are; hand-written context markdown is an efficiency lever, not a success-rate lever.**

> EVIDENCE GAP
>
> Knowledge-graph-for-code at enterprise scale is evidenced by essentially one company, via a conference talk. Treat it as promising and single-sourced.

8.2 Agent Runtime and Sandboxing

The most significant vendor-neutral development is a Kubernetes custom resource and controller for isolated, stateful, singleton agent workloads, defining a sandbox, a template, a claim, and a warm pool of pre-provisioned sandboxes to cut allocation latency, with isolation delegated to gVisor, Kata Containers, or Firecracker microVMs.[191] It is pre-1.0—real, governed, and early.

The most detailed published isolation guidance defines a ladder from a sandboxed shell tool, through a whole-process sandbox runtime, dev containers, custom containers, virtual machines, and hosted environments.[192] Three operationally load-bearing details from it belong in any platform design.

- **A per-command shell sandbox does not cover tool-protocol servers or hooks**, which run unconstrained on the host. Unattended runs require a whole-process boundary.
- **The runtime denies writes to git hooks, git config, tool-protocol manifests, agent command and subagent definitions, and shell startup files by default**—because a sandboxed session that can write those can persist configuration that runs *unsandboxed* next launch. This is Section 2.2's configuration-as-execution-surface finding arriving from the runtime side.
- On at least one platform the deny list is built once at launch and does not cover repositories created mid-session, which is a real gap for agents that clone.

The published operational benefit is one number and it is worth citing: sandboxing reduced permission prompts by 84 percent in internal usage.[192] Sandboxing is not only a security control; it is what makes unattended operation tolerable.

Two enterprise patterns are documented. Pre-provisioned pods with pre-indexed code so agent environments start in seconds.[103] And a git worktree per agent session, letting dozens of agents work concurrently on the same codebase without interference—lighter than VM isolation and suited to trusted first-party code.[100]

> EVIDENCE GAP
>
> **Cost is the gap.** Published per-vCPU-hour list prices for managed sandbox providers span roughly $0.05 to $0.13, with memory billed separately.[201] **No enterprise has published its actual annual spend on agent sandbox compute, or the ratio of sandbox compute to token spend.** Model it; do not cite it.

8.3 Tool and Capability Exposure

This is the best-evidenced layer, with converging independent implementations.

The transferable idea is auto-generation rather than curation: tool definitions generated automatically from existing interface definition language—protobuf and Thrift—across more than 10,000 internal services, eliminating manual registration, behind a single governed gateway exposing more than a thousand tools that enforces authorization, redaction, and continuous scanning on every call, and maintains a hard tiered-trust distinction between internal and third-party servers.[103] The reported effect of response shaping at that gateway is a **greater than 40 percent reduction in token usage across the agent fleet**—a larger and more reliable lever than model routing, and one almost nobody discusses.

An open-source reference architecture exists and is implementable: a reverse proxy fronting a registry API with vector and lexical hybrid discovery, **scope-based access control enforced at both discovery time and invocation time** so different users see different asset sets, scanning at registration, audit logging of who called which tool with which parameters, and federation with external registries.[194] Elsewhere, an internal developer portal exposes its catalog, scorecards, and scaffolder as registered actions through a single endpoint, with an OAuth flow removing hardcoded secrets—the concrete answer to what an IDP should expose to agents.[195]

The community protocol registry remains in preview with its API frozen, so do not build a dependency on it being generally available.[206]

> EVIDENCE GAP
>
> **Least evidenced sub-topic:** tool approval workflows. Described normatively and implemented in the open-source registry above, but no organization has published how its approval process actually operates or what its throughput and rejection rates are.

8.4 Agent Identity at Platform Scale

There is a converged, standards-based answer here, and one production implementation with published latency.

The implementation combines an agent registry as the source of truth for agent-to-workload association, a mesh in which agents carry signed tokens, a security token service issuing **short-lived, audience-scoped, single-hop tokens for every hop with the actor chain embedded**, based on OAuth 2.0 token exchange; a tool gateway as the policy enforcement point, and workload identity documents fetched via SPIFFE/SPIRE before tokens are requested. Reported: token exchange at P99 below 40 milliseconds, in use by thousands of internal agents.[193]

> The single-hop, audience-scoped, actor-chain-embedded token is the design decision to copy. It solves the problem Section 17's identity domain names as unresolvable in most estates: reconstructing who authorized what across an agent-to-agent boundary.

The scale of the underlying problem is documented: non-human identities outnumber human users at ratios between 45:1 and 144:1, only 15 percent of organizations feel confident preventing non-human-identity attacks, 51 percent report no clear ownership of AI identities, 47 percent of such identities are unchanged for over a year, one in twenty carries full administrative privilege, and only 20 percent have a formal key-offboarding process.[116] Those figures are aggregated from third-party industry reports rather than a single fielded survey and should be attributed accordingly. Platform-native agent identity is arriving commercially—one major identity provider now offers agent identities with blueprint templates carrying parent-child relationships, lifecycle management, and cross-platform federation—though its documentation states no formal general-availability date and extending existing security features to agents requires an additional license.[205]

8.5 Evaluation Infrastructure

Flag this plainly: almost everything published about enterprise evaluation infrastructure is vendor marketing. One substantive enterprise build exists in the public record.

That build is instructive because its golden set is generated continuously from production outcomes rather than hand-curated: an automated acceptance-rate pipeline comparing every agent suggestion against the merged codebase, measured across 5,230 sampled review comments on 1,727 pull requests with 90.1 percent evaluated at high confidence. Overall suggestion acceptance 63.9 percent, decomposed by category—logic errors 80 percent, bug fixes 58.1 percent, refactoring 43.5 percent, security fixes 40.6 percent. Architecturally it runs multiple independent reviewer agents on different models, treats **cross-model convergence as strong evidence**, and routes unique findings to separate verification.[202]

Two design ideas there are worth more than the numbers: **derive the golden set from production merge outcomes** so it never goes stale, and **use cross-model agreement as a confidence signal** rather than trusting a single judge.

> EVIDENCE GAP
>
> There is **no published data at all** on eval result storage and trending at enterprise scale, what fraction of organizations gate CI on evals, golden-set sizing or maintenance cost, or the cost of running evals. “CI-integrated evals with stored, trended results” is a prescription that commentary asserts and almost no organization has published evidence of operating.

8.6 Cost, Quota, and Capacity

Spend is well documented; control is not.

Total per-developer cost for teams mixing inline and agentic tools runs $200 to $600 per month including tokens, against seat list prices of roughly $19 to $60—a four- to ten-fold multiple on the procurement line item—with governance infrastructure a further $50,000 to $250,000 annually.[40] One organization’s AI-related costs rose sixfold since 2024, and it now caps every employee at $1,500 in monthly token spend per AI coding tool; at two tools that is roughly $36,000 per engineer per year.[102,196] Across a five-hundred-organization panel, median annual AI spend rose from about $1,500 to $44,000.[104]

Documented control mechanisms are simple and effective: a central gateway through which all model calls route, with a sub-100-millisecond governance latency budget at roughly 100 million requests a day; per-person daily spend alerts; and a commercial quota system built on request multipliers by model with monthly allowances and metered overage.[103,204,200]

> EVIDENCE GAP
>
> **Model routing research is strong in benchmark settings and unproven in production.** A survey reports systems achieving 84 percent cost savings with competitive accuracy, and 97 percent of a frontier model's quality at 24 percent of cost under time constraints.[199] **No enterprise has published measured savings from production model routing or caching.** The one large published fleet-level saving came from response shaping at the tool gateway, not from routing—a different and arguably more useful lever.

8.7 Observability

The transport is settling and the semantics are not. All generative-AI semantic convention attributes, spans, metrics, and events carry development status with none stable; the conventions were extracted into a dedicated repository in June 2026 and that repository has no tagged release.[90] Any organization standardizing agent telemetry today is building on a moving spec and should expect to re-instrument. Emit it anyway, and treat attribute names as a versioned contract with your own pipeline.

Published examples of what platform teams actually instrument are fragments: latency, suggestion acceptance rate, completion rate, and **provider failures**—the last having no analogue in conventional monitoring;[202] a governance-latency budget on the model gateway;[103] and **session transcripts flowing back into the service catalog after each agent session**, which is architecturally distinct from span-based tracing and gives org-wide visibility into what agents did and for whom.[195]

> EVIDENCE GAP
>
> **No enterprise has published an agent-fleet observability architecture**—no data on trace volume, retention, storage cost, session sampling strategy, or how session traces join to code outcomes. This is the least-evidenced layer alongside evaluation infrastructure.

8.8 Who Builds It

Org design here is essentially unevidenced, and the two real data points are worth knowing precisely because they are so far apart from the speculation.

One organization runs its model gateway and associated AI platform with **approximately six engineers** against an engineering organization of thousands.[204] Another names a platform surface spanning model infrastructure, tool gateway, model gateway, agent builder, agent studio, background agents, automated review, and test generation—and has not published the headcount running it.[102]

The most useful org-design model available is reasoned rather than measured: stream-aligned teams own dynamic context, business intent, and domain guardrails; the platform team owns systemic context, organizational guardrails, and tooling as self-service; enabling teams are explicitly temporary and their shrinkage is the success signal; and a guardrail appearing in three teams graduates from local to platform capability. It sets a viability threshold of three to five parallel application teams before a shared agent platform pays for itself.[212]

> EVIDENCE GAP
>
> **There is no credible published headcount, salary, or prevalence data for an "agent operations" function.** Job postings exist. Anyone claiming the role has emerged is extrapolating from job ads.

8.9 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
PLAT-1	**Context substrate with entitlement-preserving retrieval**	Retrieval scoped to the invoking principal	Enforced at the data layer; verified by adversarial retrieval test; catalog and ownership topology available to agents
PLAT-2	**Agent runtime isolated at process scope with default-deny egress**	Containerized, non-privileged, allowlisted egress	Whole-process isolation covering tool servers and hooks; ephemeral and destroyed per task; warm pools for latency
PLAT-3	**Agent runtime denies writes to persistence-bearing configuration**	Configured	Enforced and verified, including repositories created mid-session
PLAT-4	**Tools exposed through a governed gateway, not point integrations**	Gateway exists; tools inventoried	Auto-generated from service definitions; authorization and redaction enforced per call; internal and third-party trust tiers separated
PLAT-5	**Agent identity registry with short-lived, audience-scoped, single-hop tokens**	Unique identity per deployment; vaulted credentials	Token exchange with embedded actor chain; workload identity attested; delegation reconstructable after the fact
PLAT-6	**Evaluation infrastructure operated as platform, not per team**	Registry of corpora, metrics, thresholds, owners	Golden sets derived from production outcomes; cross-model agreement used as a confidence signal; results stored and trended

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
PLAT-7	**All model calls routed through a governed gateway**	Central gateway in place	Per-team quotas enforced; anomaly alerting; response shaping to control token cost
PLAT-8	**Cost attributed per agent, per task, and per team**	Attribution beyond the billing account	Enforced ceilings with automated halt; unit cost per accepted change reported
PLAT-9	**Agent telemetry emitted in a versioned schema**	Sessions traced and retained	Schema pinned and versioned; session transcripts joined to code outcomes; generated-to-analyzed ratio measured
PLAT-10	**Named platform team with funded ownership**	Ownership assigned	Staffed team with a published service model, guardrail graduation process, and measured internal adoption

8.10 Evidence to Request

- An adversarial retrieval test showing the substrate refuses content the invoking human cannot see.
- A demonstration that an agent cannot write to git hooks, tool manifests, or shell startup files.
- A delegation chain trace across an agent-to-agent boundary, reconstructed after the fact.
- Cost per accepted change, attributed to a team.
- The generated-to-analyzed ratio for agent telemetry.
- The platform team's headcount against the engineering organization, and what it has graduated from local guardrail to platform capability in the last two quarters.

8.11 Failure Modes

- **Buying tools instead of building substrate.** The published outcome differences between organizations are platform differences, not tool differences.
- **Point integrations instead of a gateway.** Every tool-protocol server stood up by a team to make an agent useful is an ungoverned credentialed path into the estate.
- **Sandboxing the shell and not the process.** Tool servers and hooks run outside the per-command boundary, which is exactly where persistence is established.
- **Retrieval scoped by prompt.** "Only look at what the user can see" is an instruction. Entitlement enforcement belongs at the data layer.

- **Evaluation infrastructure owned per team.** Metric definitions diverge, results are not comparable, and nothing trends.
- **Cost visible only at the account level.** It cannot detect a looping agent, support a chargeback conversation, or inform a tier decision.

09 The Human Operating Model

OBJECTIVE

Define what people do when agents produce most of the code—and protect the capabilities the rest of this framework depends on.

Every control in Part IV assumes a human who can specify precisely, design an oracle, review a high-risk change, and recognize when an agent's account of its own work is wrong. Those capabilities are not automatic, they are currently being depleted, and no part of this transition works without them.

9.1 What the Work Becomes

The functional shift is empirically documented, though as cognitive redefinition rather than as job titles. A study of 319 knowledge workers across 936 real usage instances identifies three effort transitions: **information gathering becomes information verification; problem-solving becomes response integration; task execution becomes task stewardship.**[209] That is the role change, measured. Job titles have not caught up, and the only emerging title with hard posting data (922 postings, a fivefold year-over-year increase) is a rounding error against a labor market of millions.[220]

The honest framing for that market: aggregate software development postings sit roughly 25 percent below their February 2020 baseline.[220] Role composition is shifting inside a contracting pool, which is a materially different situation from roles evolving inside a growing one.

Concretely, at Stage 2 and beyond, human engineering time redistributes toward five activities.

- **Specification.** The highest-leverage work in an agentic organization and the least rewarded by current career ladders. Section 6.1 puts specification completeness as the second-strongest predictor of delegation success; the measured jump from 38.1 to 69 percent came from preparation, not models.[105]
- **Oracle design.** Deciding what "correct" means and building the mechanism that checks it, independent of the agent. This is a distinct skill from writing tests and most organizations have nobody explicitly accountable for it.
- **Architecture and constraint definition.** Including the machine-checkable form, per Section 7.2's finding that up to 76 percent of what review catches is in principle automatable and the shortfall is project-specific rules nobody has codified.
- **Review at the boundary.** Not all review—the high-risk, design-bearing, intent-bearing subset that Section 7.7 says human review is actually good at.
- **Escalation arbitration and incident judgment.** Including the specific competence of not believing an agent's self-report. In the best-documented agent incident the agent falsely reported that rollback was impossible when it was not.[14]

9.2 The Ratio Question

Executives want a number. **There is no published empirical data establishing how many concurrent agents an engineer can effectively supervise—not from a vendor, not from an enterprise, not from academia.** The concept is widely discussed and entirely unevidenced.

The term itself originates in a vendor survey that coined "human-agent ratio," named three states (too few agents, too many overwhelming human judgment capacity, and optimal) and **supplied no formula, benchmark, or tested figure**, explicitly treating the optimum as task- and organization-dependent.[217] Vendor frameworks that assert ratios cite real research for context and nothing for their own numbers, labeling their worked examples as illustrative parameters to be replaced with measured data.

The strongest empirical data point on parallel supervision is a single experiment with one human: sixteen parallel agents, roughly 2,000 sessions, just under two weeks, and about $20,000, producing a 100,000-line compiler that built a bootable kernel across three architectures.[216] The instructive detail is where the human effort went—into designing the test harness and environmental feedback loops, and then "(mostly) walking away." **Supervision capacity was not spent on individual agent decisions; it was spent on building the oracle.** That is the correct generalization, and it is the only one the evidence supports.

The nearest thing to a real operating figure is expressed as scope rather than agent count: a single engineer running fleet-wide migrations that previously required hundreds of teams, with one migration completing in three days rather than weeks.[101]

> WHAT TO MEASURE INSTEAD
>
> Ratio is not yet a defined unit. Absorption capacity is. Track review hours required against senior engineering hours available; changes merged without human review; review duration and approval rate; and the escalation rate per agent-week. Those are measurable today, they bind directly, and they answer the question the ratio was a proxy for.

9.3 Restructuring Review

Uniform review does not survive the volume, and Section 3.3 shows organizations discovering this by failing at it. Four adjustments have some published basis.

- **Risk-tier the change rather than the author.** Security-relevant, authorization-bearing, configuration, dependency, infrastructure, and public-interface changes receive mandatory senior review regardless of size. Reviewers already behave this way—they take a median 3.92 hours on security-relevant agent pull requests against 0.11 hours otherwise.[171] Formalize what is already happening.
- **Auto-merge what is provably safe and concentrate humans at the boundary.** The one documented enterprise practice at scale.[101] This is only defensible when the oracle is strong enough to define "provably," which is why Section 4.2 gates it behind Stage 3 entry criteria.
- **Prefer collaboration over supervision on hard changes.** When humans committed directly into an agent's pull request rather than reviewing and returning it, success rose from 55 to 86 percent.[105] That is a large effect and it argues against a pure reviewer posture on complex work.
- **Push the mechanical into automation and keep the conceptual.** The 76 percent figure from Section 7.2 is the roadmap, and codified house rules are the first target.

> EVIDENCE GAP
>
> The uncomfortable open question: reviewers are currently *not* spending measurably longer on agent-authored pull requests despite lower acceptance rates, and the study observing this cannot distinguish efficiency from insufficient scrutiny.[121] No published study measures rubber-stamping rates in agent pull request review. Instrument your own.

9.4 The Junior Pipeline

This is the hardest problem in the framework and the one with the most consequential evidence. It is also where two credible datasets genuinely disagree, and this section presents both rather than harmonizing them.

The evidence for harm

Administrative payroll microdata covering a balanced panel of 3.5 to 5 million employees per month from January 2021 to June 2026 finds workers aged 22 to 25 in AI-exposed occupations sitting **19 percent below** where they would be had they kept pace with less-exposed peers, with employment in the two most-exposed quintiles falling roughly 11 percent while the three least-exposed grew roughly 10 percent.[207] The shortfall is widening—15 percent in July 2025, 19 percent by June 2026—and, critically, **the mechanism is reduced hiring rather than increased separations.** Firms are not firing juniors; they are not opening the door. The authors have separately addressed the interest-rate counterargument with firm-time fixed effects showing the decline becoming significant only after 2024.[207]

The evidence against the AI attribution

A tech-specific talent dataset finds new-graduate hiring down roughly 65 percent at major technology companies and 76 percent at early-stage startups against 2019—while stating that "the long-feared 'AI Code Apocalypse' has failed to materialize" in engineering, and assigning primary causation to the end of the zero-interest-rate era rather than to automation.[208]

Both agree completely that the junior door is closing. They disagree on why. Present it that way.

THE BIND, STATED PRECISELY

The largest measured productivity gains from AI assistance go to exactly the cohort firms have stopped hiring: 21 to 40 percent for junior and short-tenure developers against 7 to 16 percent for seniors.[27] An organization optimizing away its junior pipeline is optimizing away its highest-leverage adoption cohort while creating no mechanism to replace senior judgment as it retires—and senior judgment is the input every control in Part IV consumes.

Two firms are betting the other way, one tripling entry-level hiring and one adding a thousand graduates and interns, with the only public cost-based argument being a roughly 30 percent wage premium to buy mid-level talent rather than grow it.[221,215] Both are executive claims without outcome data.

EVIDENCE GAP

And no published study measures skill formation in juniors trained primarily under agentic workflows. The "how do seniors get made" question has strong indirect evidence and zero direct longitudinal evidence. Any organization going all-in should be running that experiment deliberately and instrumenting it, because nobody else has.

9.5 Skill Formation and Automation Complacency

The directly relevant measurement: a randomized trial of fifty-two engineers found AI-assisted participants finishing marginally faster and scoring **50 percent on a comprehension assessment against 67 percent** for those who wrote the code by hand.[18] The usage-pattern breakdown is the actionable part. Patterns averaging below 40 percent comprehension were delegation-style, asking the model to solve rather than to explain. Patterns at or above 65 percent involved generating and then interrogating, requesting explanation alongside code, or asking conceptual questions and writing the code independently.

> **The same tool produces a 25-point comprehension spread depending on how it is used.** That makes usage pattern an organizational design variable, not a matter of individual preference. It is also vendor-affiliated research with a small sample measured immediately rather than longitudinally, and it should be cited with those limits.

The mature analogue is the aviation and human-factors literature on automation complacency, where the foundational synthesis establishes that complacency and automation bias are the same underlying phenomenon with attention allocation at its center, rather than two separate failure modes.[210] The transfer to agent supervision is direct: a reviewer whose attention is allocated away from a task by reliable automation does not become a worse reviewer when the automation fails—they become a slower one to notice.

The practical implications are unglamorous. Vary what humans verify so attention does not settle. Seed the review stream with known defects periodically and measure catch rate—this is the approval-integrity probe in Section 20. And require an explanatory record for non-trivial agent-authored change at the time of authorship rather than at the time of confusion, because comprehension debt is invisible until someone needs to change the code.

9.6 Team Topology

The most credible org-design position holds that organizational maturity rather than model capability determines outcomes, reframing team structure as “infrastructure for agency” with four claims worth carrying:

- **bounded agency**, where authority to act is intentionally constrained by guardrails so delegated initiative remains governable; a
- **security asymmetry** question about why organizations grant agents data access they would never permit a human;
- **cognitive load transferring to agents**, so that agents “begin to lose coherence or hallucinate when they operate outside their defined boundaries” exactly as humans do when their capacity is exceeded; and
- **stewardship rather than ownership** as the framing for long-lived systems.[212]

That last set is more than analogy. If agent coherence degrades outside bounded scope—and Sections 5.8 and 7.4 say it does, by measured amounts that worsen with scope—then the Team Topologies cognitive-load principle applies to agents as first-class team members, and scope discipline is a correctness control rather than only a security one.

The same source points to an innovation-and-practices enabling team pattern, and observes that some organizations use **"friendly FOMO" rather than mandates** to drive adoption. Section 9.7 explains why that distinction matters more than it sounds.

> EVIDENCE GAP
>
> **What is not published:** no credible data exists on span-of-control changes, team size changes, or manager-to-IC ratio changes in AI-native engineering organizations. The claim that teams get smaller with the same output is entirely untested.

9.7 What Happened at Companies That Mandated It

Four public arcs, and the pattern across them is instructive.

A hard mandate with no reversal.

One retailer's memo made "reflexive AI usage" a baseline expectation, required teams to demonstrate they could not accomplish work with AI before requesting headcount, and added AI usage to performance review.[223] No reversal has been reported. Also no published outcome data—no headcount trajectory, no output measurement, nothing validating the headcount gate.

A mandate walked back three times.

Another company's AI-first memo triggered a backlash that produced two clarifications and then, in April 2026, a substantive reversal: performance evaluation reverted to outcomes rather than AI usage, with the chief executive stating that if AI cannot help with a job, "I'm not going to force you to do that."[213] His own diagnosis of the error is the most useful line in this literature: the policy felt to employees like being measured on tool use rather than on results.

A tooling mandate enforced by firing.

A third mandated onboarding to specific tools within a week and dismissed engineers who had not complied without good reason. The number fired was never disclosed, the stated targets were never confirmed as met, and no independent verification of the reported AI-authored code percentage exists.[222]

The best-documented reversal, and the most misreported.

A fourth automated frontline customer service, publicly claiming work equivalent to 700 agents and a projected $40 million profit improvement, then began rehiring humans a year later, with its chief executive conceding "lower quality" output.[214] The accurate arc is not "it reversed and failed." Aggressive automation held on cost and revenue and failed on quality and brand; the correction was a **re-segmentation**, automation for volume and humans for escalation and brand assurance, at a permanently smaller headcount.

THREE LESSONS

Mandating tool usage measures the wrong thing and is the specific policy that got reversed. Percentage of code written by AI is an input metric no company defines consistently and none has independently verified. And re-segmentation, not reversal, is the mature correction—which is exactly what the work-class taxonomy in Section 6 formalizes in advance rather than after a public retreat.

9.8 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
HUM-1	**Absorption capacity measured and governing merge rate**	Review hours required against available reported	Automatic merge-rate reduction when the ratio breaches threshold
HUM-2	**Review risk-tiered, with mandatory senior review on high-risk paths**	Path-based requirements configured	Enforced by code owners; exceptions documented with expiry
HUM-3	**Approval integrity measured and probed**	Approval rate and review duration reported	Periodic seeded-defect probe with measured catch rate; degradation reduces autonomy tier
HUM-4	**Named accountable human for every A2+ deployment**	Named individual, not a rotation	Ownership verified in the registry; departures trigger reassignment or decommission
HUM-5	**Specification and oracle design recognized as accountable roles**	Responsibility assigned	Reflected in career ladder and performance expectations; staffed explicitly
HUM-6	**Skill formation managed as a delivery risk**	Usage patterns discussed	Comprehension-preserving usage patterns encouraged and measured; junior pipeline maintained deliberately
HUM-7	**Every subsystem has an owner who can explain it**	Ownership assigned	Explanatory records required for non-trivial agent-authored change at authorship time
HUM-8	**Adoption driven by enablement rather than usage mandates**	Policy states outcomes, not tool usage	Performance measured on outcomes; AI usage never a performance criterion
HUM-9	**Agent self-report treated as evidence requiring verification**	Runbook states it	Exercised in incident rehearsal

9.9 Evidence to Request

- The absorption ratio, trended, with the source of its review-hour estimate.
- Results of the most recent seeded-defect probe, with catch rate and median review time.
- Tenure and attrition data for senior engineers, read alongside review load—the two together are the early warning for depleting the capability everything else depends on.
- Entry-level hiring and internal promotion rates over eight quarters.
- A named owner for a randomly chosen subsystem, and a five-minute explanation of how it works.

9.10 Failure Modes

- **Measuring engineers on AI usage.** It is the one policy in this literature with a documented public reversal, and it inverts the relationship between tool and outcome.
- **Treating "percentage of code written by AI" as a goal.** Undefined, unverified, and an input metric masquerading as an outcome.
- **Cutting the junior pipeline while depending on senior judgment.** The cohort with the largest measured gains is the cohort being cut, and there is no published mechanism for producing seniors without it.
- **Uniform review as policy and waived review in practice.** Both failure modes appear in the same dataset; the merge record looks identical either way.
- **Assuming a supervision ratio.** Nobody has one. Measure absorption instead.
- **Believing the agent's account of the incident.** It is the single documented instance where an agent's self-report was materially false during a live incident, and it is enough to warrant a runbook line.

PART III

The Lifecycle

The seven stages, as they change. Each carries the same structure: what changes, elements to adopt, controls at a Level 2 minimum bar and a Level 3 enforced state, evidence an auditor can request, and failure modes.

10 Stage One — Planning

OBJECTIVE

Establish the problem, the business case, the scope, the feasibility, and the resourcing—with AI participation treated as an explicit input to each rather than an unpriced assumption.

10.1 What Changes at This Stage

Planning is where the agentic SDLC most often goes wrong, and it goes wrong quietly, because the error is an omission rather than a mistake. Organizations plan the delivery and do not plan the absorption.

The evidence in Section 3.3 is unambiguous that generation capacity rises faster than review capacity. A plan that assumes a 30 percent throughput improvement and staffs accordingly, without provisioning for a several-fold increase in review load, is a plan to accumulate unreviewed change. The Faros telemetry finding of pull requests merged without review rising 31.3 percent is not a description of bad engineers. It is the arithmetic consequence of raising the numerator and holding the denominator constant.

Three further planning assumptions require revision.

Cost is no longer a seat license. Per-seat list pricing for AI development tooling ranges roughly from $19 to $100 per developer per month, but observed total cost per engineer, combining license and consumption, commonly runs $200 to $600 per month for teams using both inline and agentic tooling—a four- to ten-fold multiple on the line item that appears in the procurement request.[40] The market has been shifting from seat-based to usage-based pricing, which converts a fixed cost into a variable one and moves it from a predictable annual commitment into something that behaves like cloud spend.[41] The FinOps Foundation's 2026 survey found 98 percent of organizations now managing AI spend, against 31 percent two years earlier, and ranked AI cost management as the single skillset teams most need to develop.[42] Plan for AI cost as consumption, with an owner, a forecast, and an anomaly response—not as a software license.

Skill formation is a planning variable. The randomized trial cited in Section 2 found comprehension outcomes varying by usage pattern rather than by tool: patterns averaging below 40 percent on comprehension were delegation-style, asking the model to solve rather than to explain, while patterns at or above 65 percent involved generating and then interrogating, or requesting explanation alongside code.[18] If a plan's staffing model depends on junior engineers becoming senior engineers, the usage pattern the organization encourages is a delivery risk, not a cultural preference.

Upstream contribution may be constrained. Open source projects have diverged sharply on AI-authored contributions, and enterprises with upstream dependencies should know which posture their dependencies hold before planning work that requires upstream acceptance. The Linux kernel, Fedora, LLVM, the OpenInfra Foundation, OpenTelemetry, and Rocky Linux accept AI-assisted contributions subject to disclosure via an `Assisted-by:` commit trailer, with the kernel's policy stating flatly that "AI agents MUST NOT add Signed-off-by tags. Only humans can legally certify the Developer Certificate of Origin."[43] QEMU declines contributions believed to contain AI-generated content; the Gentoo Council voted unanimously to forbid contributions produced with natural-language AI tools; NetBSD presumes such code tainted and requires explicit core approval.[44] These are incompatible governance postures, and a roadmap that assumes an upstream patch will be accepted may be planning against a project that will not take it.

10.2 Elements to Adopt

An AI participation statement, per initiative. Before work begins, record which stages will involve AI participation, at which autonomy tier, and what that implies for review capacity, cost, and skill development. This is a short document, not a governance ceremony. Its value is that it forces the absorption question to be answered in writing at the point where staffing is still adjustable.

Absorption capacity modeling. Estimate review hours required per unit of delivered change under the planned AI participation, against senior engineering hours actually available. Where the model shows a deficit, the plan must resolve it—by lowering the autonomy tier, by investing in automated verification that reduces per-change review cost, by hiring, or by accepting a lower merge rate. Deferring the question to the team is how unreviewed merges happen.

Consumption forecast and unit economics. Establish cost per task, not cost per token, as the planning metric. Per-token prices for a fixed capability level have fallen rapidly—roughly fortyfold per year for the price to match a given performance threshold, though with a spread of ninefold to nine-hundredfold depending on task type, and that analysis dates from early 2025 and should be treated as such.[45] Agentic workloads consume dramatically more tokens per unit of work than conversational use, so falling unit prices coexist with rising bills. Forecast the bill.

Regulatory scoping. Determine at planning time which regimes apply to the system, and whether AI participation in its construction changes the answer. The EU Cyber Resilience Act's reporting obligations for actively exploited vulnerabilities and severe incidents apply from September 11, 2026, on a cadence of a twenty-four-hour early warning, a seventy-two-hour notification, and a final report within fourteen days for an actively exploited vulnerability or within one month for a severe incident, with full application from December 11, 2027.[46] It is the first regime to make secure development lifecycle practice a market access condition rather than a procurement condition, and it does not care which agent wrote the code.

Vendor and model dependency scoping. Identify which model providers the initiative will depend on and confirm their deprecation policies before the architecture assumes them. Published notice commitments differ materially: one major provider commits to at least six months' notice for generally available models and as little as two weeks for preview models; another commits to at least sixty days but publishes forward-dated "not sooner than" floors that the first does not.[47,48] Neither addresses behavioral change within a pinned snapshot, which is the risk Section 16 covers.

10.3 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
PLN-1	**AI participation statement produced per initiative**	Written, reviewed at planning sign-off	Required field in the initiative record; absent statement blocks funding release
PLN-2	**Absorption capacity modeled against planned change volume**	Estimated and documented	Modeled from actual historical review telemetry; deficit triggers automatic tier reduction
PLN-3	**AI consumption forecast with named cost owner**	Annual forecast at initiative level	Per-team budget with anomaly alerting and enforced ceilings
PLN-4	**Autonomy tier proposed and approved at the correct authority**	Tier recorded in the initiative record	Tier assignment integrated with repository and credential provisioning
PLN-5	**Regulatory and contractual scope determined, including AI-specific obligations**	Documented mapping	Mapping maintained as a living artifact with named owner and review cadence
PLN-6	**Upstream contribution constraints identified for affected dependencies**	Checked for primary dependencies	Automated policy check against a maintained register of upstream AI policies
PLN-7	**Skill formation impact considered in staffing model**	Discussed and recorded	Usage-pattern telemetry informs team composition and enablement investment

10.4 Evidence to Request

- The initiative record containing the AI participation statement and approved tier.
- The absorption model with its inputs and the source of its review-hour estimates.
- The consumption forecast alongside the previous period's actuals, which is where forecasts are revealed as aspirational.
- Approval records showing the tier was signed at the required authority level rather than assumed.

10.5 Failure Modes

- **Planning the acceleration and not the consequence.** The most common failure, and the one with the longest tail. It manifests three to six months later as a review backlog, a rising unreviewed merge rate, and an incident rate that nobody connects back to a planning assumption.

- **Treating AI cost as a license line.** Procurement approves a seat count, finance budgets a seat count, and the consumption bill arrives against a different cost center with no owner. This resolves badly and usually loudly.
- **Assigning a tier by default.** Nobody decides that the coding agent operates at A3; it simply arrives configured that way, and the first time anyone examines the question is during an incident review.
- **Assuming the pilot's economics.** Pilots run with enthusiastic early adopters on greenfield work—the population and task type where measured gains are largest. Extrapolating those numbers to maintenance work in a mature codebase, where the strongest available RCT found a slowdown, produces a business case that cannot be met.[33]

11 Stage Two — Requirements Analysis

OBJECTIVE

Determine what the system must do, in a form precise enough that both human engineers and autonomous agents can build against it and both automated and human verification can test against it.

11.1 What Changes at This Stage

Requirements have always been the highest-leverage stage and the most under-invested. Agentic delivery raises the leverage further, in both directions.

The upside is real and underappreciated: agents are unusually good at building exactly what a specification says. The downside is the same sentence. A human engineer receiving an ambiguous requirement typically resolves the ambiguity by asking, by inferring from context, or by noticing that the requirement contradicts something they know about the system. An agent resolves it by generating something plausible. Specification defects that were previously absorbed by human judgment now propagate directly into implementation, at machine speed, across multiple work streams simultaneously.

This inverts a long-standing economic assumption. For thirty years the argument for lightweight requirements was that specification effort was expensive relative to the cost of iterating in code. When implementation cost falls sharply and specification cost does not, the ratio flips: specification becomes the expensive, scarce, high-value activity, and it is where senior engineering judgment now earns the most.

A second change concerns requirements *for* AI-backed capability. A functional requirement for a deterministic component can be stated as a behavior. A functional requirement for a model-backed component cannot; it must be stated as a distribution with an acceptance threshold, a measurement method, and a defined behavior under failure. “The system summarizes the document accurately” is not a requirement. “The system produces summaries scoring at or above 0.8 on the defined rubric for at least 95 percent of the evaluation corpus, with a defined refusal path below a confidence threshold, measured against a held-out set refreshed quarterly” is a requirement, and it is one an evaluation harness can test.

11.2 Elements to Adopt

Machine-legible acceptance criteria. Every requirement should carry acceptance criteria expressed in a form that can drive an automated check. This does not mandate a formal specification language; structured natural language with explicit preconditions, postconditions, and error behavior is sufficient and far more likely to be maintained. The test is whether an agent given only the criteria, and no access to the person who wrote them, would build the right thing.

Explicit non-functional requirements, stated numerically. The accessibility evidence in Section 3.5 is the clearest available demonstration of what happens to quality attributes that are not stated as requirements: they degrade, at population scale, without anyone deciding to degrade them.[37] Performance budgets, accessibility conformance targets, security requirements by CWE class, resource ceilings, and operability requirements should be stated as testable numbers at this stage, because nothing downstream will introduce them.

Security requirements specified by vulnerability class, not by aspiration. Given measured pass rates of 15 percent on cross-site scripting and 13 percent on log injection against 82 percent on SQL injection, the classes requiring explicit specification and explicit verification are known in advance.[15] A requirements process that says "the system shall be secure" transfers the whole problem to a code review that the evidence says is already saturated. Name the classes, state the controls, and make them acceptance criteria.

Provenance and attribution requirements. Where the system will be distributed, embedded in a regulated product, or contributed upstream, requirements for artifact provenance, AI-authorship disclosure, and bill-of-materials content belong here rather than being discovered at release. This is also where licensing exposure is cheapest to address, and the exposure is real: a study generating more than seventy thousand method implementations against copyleft-licensed source found 1.48 percent of methods showing mean similarity above 0.7 to licensed code, rising with larger context—the finding most directly relevant to agents, which operate with far larger context windows than autocomplete.[49] The vendor's own recitation research, dated but the only rigorous first-party measurement available, put verbatim recitation at roughly 0.009 percent of suggestions in 2021-era conditions.[50] The litigation position remains unsettled: the class action over AI code generation was argued before the Ninth Circuit on February 11, 2026 on whether the DMCA's copyright management information provision contains an identicality requirement, and no opinion had issued as of late August 2026.[51] A ruling on that question would be the most consequential single development for AI code intellectual property risk, so verify the docket rather than relying on this paragraph.

Requirements for the agents themselves. Where agents will participate at A2 or above, their scope, permitted tools, and expected behavior are requirements and should be specified and reviewed as such—not configured later by whoever sets up the repository.

Traceability that survives generation. Requirement-to-implementation traceability was historically maintained by the humans who did both. When implementation is generated, traceability must be explicit: a requirement identifier carried into the task, into the branch, into the commit, and into the test. This is unglamorous and it is what makes impact analysis possible eighteen months later.

11.3 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
REQ-1	**Acceptance criteria machine-legible for every requirement**	Structured criteria present and reviewed	Criteria drive automated verification; requirements without executable criteria cannot be scheduled

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
REQ-2	**Non-functional requirements stated as testable numbers**	Performance, accessibility, and resource targets specified	Targets enforced as pipeline checks with defined failure behavior
REQ-3	**Security requirements specified by vulnerability class**	Named classes with required controls	Class-specific verification bound to acceptance; high-failure-rate classes carry mandatory checks
REQ-4	**Probabilistic requirements expressed as thresholds with a measurement method**	Threshold, corpus, and metric defined	Evaluation harness derived directly from the requirement; corpus refresh scheduled
REQ-5	**Provenance, attribution, and licensing requirements captured**	Documented per initiative	Enforced at build; artifacts without required provenance fail release
REQ-6	**Agent scope specified as a reviewed requirement**	Scope documented for A2+ deployments	Scope expressed as enforced configuration; drift between specified and actual scope alerts
REQ-7	**Requirement-to-artifact traceability maintained**	Identifier carried into work items	Identifier carried through commit, test, and release record; automated coverage reporting

11.4 Evidence to Request

- A requirement selected at random, traced forward to its implementing commits, its tests, and the release that shipped it.
- The evaluation corpus and threshold for a model-backed requirement, with the date the corpus was last refreshed.
- The non-functional requirement set for a recently shipped feature, and the pipeline check that enforces each item.
- Where a requirement has no corresponding check, that is the finding.

11.5 Failure Modes

- **Ambiguity as an implicit delegation.** Requirements that a human would have queried are now silently resolved by a model. The resulting system is defensible against the specification and wrong against the intent, and the gap is discovered in production.
- **Quality attributes left unstated.** Accessibility, performance, and operability are assumed to be part of “doing it properly.” Under generation at volume, nothing that is not specified and checked survives.

- **Behavior specified for a probabilistic component.** A requirement written as though the model were deterministic produces a test suite that is either flaky or vacuous, and usually both in sequence.
- **Traceability abandoned as overhead.** It was already the first thing cut under delivery pressure. It is now the thing that determines whether you can answer a regulator, scope an incident, or safely delete anything.

12 Stage Three — Design

OBJECTIVE

Determine how the system will satisfy its requirements—and, distinctly, how the system will be constructed, verified, and operated in an environment where a substantial fraction of its code is generated.

12.1 What Changes at This Stage

Design acquires a second audience. Architecture has always been written for the engineers who will implement and maintain the system. It is now also read, at scale and without the ability to ask questions, by the agents that will implement it. A design that relies on tacit team knowledge, on conventions that live in people's heads, or on the implicit understanding that "we don't do it that way here," will not survive contact with a generative process that has no access to any of that.

This produces a design obligation that has no precedent: **architecture must be legible to a non-participant**. The mechanisms are not exotic—explicit module boundaries, dependency rules stated rather than implied, naming conventions written down, a documented rationale for the patterns in use, machine-readable constraints where the toolchain supports them. What is new is that the cost of illegibility is now paid continuously and automatically rather than occasionally at onboarding.

Second, the maintainability data in Section 3.6 should be read as a design finding, not merely a quality finding. Refactoring collapsing from 21 percent to 3.8 percent of changes, cross-file function calls down 35 percent, and duplication up 81 percent describe a codebase drifting toward local, duplicated, low-reuse structure.[38] Generation is locally optimal and globally indifferent: a model asked to implement a function will implement it, not notice that something similar exists three modules over. Reuse, abstraction, and consolidation are now architectural decisions that must be designed for and enforced, because they will not emerge from the generation process.

Third, design is where the blast radius of every downstream agent action is set. An agent can only do damage in proportion to what the architecture lets any single component reach. Module boundaries, credential scoping, environment separation, and the blast radius of a bad deployment are agent-safety controls, whatever else they were designed to be.

Fourth, where the system embeds model-backed capability, design must address the model as a live dependency: which provider, which pinned snapshot, what happens on deprecation, what happens on silent behavioral change, whether a fallback path exists, and whether the system degrades gracefully when the model is wrong rather than merely when it is unavailable.

12.2 Elements to Adopt

Agent-legible architecture. Write the constraints down. Module boundaries, allowed dependency directions, layering rules, naming conventions, error handling patterns, logging expectations, and the reasoning behind each. Place them where the toolchain will surface them to whatever is generating code—repository-level guidance files have become the de facto mechanism, and their significant property is that they are source-controlled, reviewable artifacts rather than platform configuration.[52] Treat them as design documents subject to design review, because that is what they are.

Enforced architectural constraints. Documentation states intent; automated architectural checks (dependency rule enforcement, layer violation detection, cyclic dependency detection, module boundary tests) establish it. Under generation at volume, the check is what holds the line. Include duplication thresholds, because duplication is now the measured failure mode.

Blast radius as a design parameter. For each component, state explicitly what an autonomous change to it could affect: which data, which systems, which customers, which credentials. Design so that the components where an agent will operate most are the components with the smallest reachable radius. This is ordinary good design; what is new is the reason to insist on it.

Environment separation that is architectural, not procedural. The single clearest lesson from documented agent incidents is that separation enforced by process fails and separation enforced by topology holds. Development environments must not hold production credentials. This is not a new principle; it is a principle whose violation is now exploited automatically rather than occasionally.

Model dependency design. For every model-backed capability, specify the pinned snapshot rather than a floating alias, the fallback behavior on unavailability, the fallback behavior on *degraded quality*, the evaluation gate that governs a version change, and the owner. Treat the model as a dependency with a version, an owner, and an expiry date—but understand that pinning bounds the risk rather than eliminating it, since providers can change API surface and parameter behavior independently of model retirement.[48]

Design for verification. Where correctness is statistical, the system must be architected so that correctness can be measured in production: capture points for inputs and outputs subject to data classification policy, correlation identifiers, the ability to replay against a different model version, and instrumented decision points. Retrofitting observability into a probabilistic system is materially harder than into a deterministic one, because the thing you need to observe is a distribution.

Tool-protocol surfaces as designed interfaces. Where agents will reach internal systems through a tool protocol, those servers are privileged integrations and belong in the architecture, with an authorization model, an owner, and a review cadence. The current specification is explicit that servers "MUST only accept tokens that are valid for use with their own resources" and "MUST NOT accept or transit any other tokens," and that possession of a state handle must not be treated as authentication.[53] Those requirements describe a confused-deputy failure mode and a session-fixation failure mode respectively, and both are design problems rather than implementation details. The protocol's governance moved to a Linux Foundation directed fund in December 2025, which is a relevant durability signal for architecture decisions that depend on it.[54]

Threat modeling that includes the toolchain. Extend threat modeling to cover the development environment itself. The relevant taxonomies are published and usable: the OWASP Top 10 for Agentic Applications, released December 2025, enumerates agent goal hijack, tool misuse and exploitation, identity and privilege abuse, agentic supply chain vulnerabilities, unexpected code execution, memory and context poisoning, insecure inter-agent communication, cascading failures, human-agent trust exploitation, and rogue agents.[55] The 2026 revision of the OWASP LLM Top 10 is the first edition to weight real incident data alongside expert voting, drawing on 7,714 collected incidents of which 6,639 were classifiable, weighted at 25 percent against 75 percent community voting, and moved excessive agency from sixth to third place on that basis.[56] MITRE ATLAS provides the adversary technique vocabulary and now includes a case study specifically covering supply chain attacks against AI coding assistants via configuration files.[57]

12.3 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
DSN-1	**Architectural constraints documented in a machine-surfaced, source-controlled artifact**	Present and reviewed	Constraint file changes require architecture review; drift from enforced checks alerts
DSN-2	**Architectural rules enforced by automated check**	Dependency and layering checks in CI	Checks blocking at merge, including duplication and reuse thresholds
DSN-3	**Blast radius documented per component**	Documented for components in agent scope	Blast radius bounded by enforced credential and network scope; verified in the running environment
DSN-4	**Environment separation enforced topologically**	No production credentials in development environments	Separation verified by automated test; violation blocks deployment
DSN-5	**Model dependencies pinned to dated snapshots with declared fallbacks**	Snapshot pinned, owner named	Version change gated on evaluation; alias use blocked at build
DSN-6	**Design for observability of probabilistic behavior**	Capture and correlation points specified	Replay capability tested; evaluation instrumentation present at design review
DSN-7	**Tool-protocol integrations reviewed as privileged interfaces**	Inventoried with named owners and authorization model	Version-pinned, signed where supported, capability drift detected automatically

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
DSN-8	**Threat model covers the development toolchain and agent surfaces**	Toolchain threat model documented	Reviewed on tooling change; mapped to a published agentic risk taxonomy

12.4 Evidence to Request

- The architectural constraints artifact, with its review history—a file that has never been reviewed is configuration, not architecture.
- The output of architectural enforcement checks over the last quarter, including violations and how they were resolved.
- A component's documented blast radius, verified against the credentials actually present in its runtime.
- The model dependency register with pinned snapshots, owners, and provider-published retirement floors.
- The toolchain threat model, with the date it was last revised against the current tool inventory.

12.5 Failure Modes

- **Architecture that lives in people's heads.** Previously a hiring and onboarding cost. Now a per-commit cost, paid automatically, forever.
- **Constraints documented but unenforced.** A guidance file with a rule that no check verifies is a suggestion. Under generation at volume, suggestions lose.
- **Reuse assumed rather than designed.** Nothing in a generative workflow rewards finding the existing implementation. Absent an enforced duplication threshold and a discoverable component catalog, duplication is the default outcome, and the data says it is the observed one.[38]
- **Tool-protocol servers as convenience integrations.** Stood up by an engineer to make an agent useful, holding broad credentials, never reviewed, invisible to the architecture. This is the most common way an agent acquires reach that nobody designed.
- **Designing for model availability rather than model correctness.** Fallback paths handle the case where the provider is down. They rarely handle the case where the model returns a confident, well-formed, wrong answer—which is the far more common failure.

13 Stage Four — Development and Implementation

OBJECTIVE

Produce code, configuration, and infrastructure definitions that meet the requirements—under conditions where a large and growing share of the artifacts are machine-generated, the development environment is itself an attack surface, and review is the binding constraint.

This is the longest section in the framework, because it is where the greatest number of controls bind and where the documented failures cluster.

13.1 What Changes at This Stage

The development environment became an execution surface. This is the most defensible technical claim available about agentic development, and it is supported by a consistent pattern across every significant disclosed vulnerability in this class. In two Cursor vulnerabilities published to the national vulnerability database in early August 2025, prompt injection rewrote a tool-protocol configuration file whose new entry executed before the user could reject the suggested edit, and a separate flaw bound approval to a server *name* rather than to configuration contents, so an attacker with repository write access could swap the command body of an already-approved entry and obtain silent execution on every subsequent launch.[19,20] In the CodeRabbit compromise disclosed in August 2025, a linter configuration file supplied in an ordinary pull request executed attacker-controlled code on the analysis host, yielding the platform's GitHub App private key and with it read and write access to approximately one million repositories.[21] The "Rules File Backdoor" technique published in March 2025 poisons assistant *instruction* files with directives concealed in zero-width joiners and bidirectional text markers, which render as blank space in review and in pull request diffs, and is now a MITRE ATLAS case study.[22,57] Claude Code vulnerabilities disclosed by external researchers in 2026 followed the same shape: untrusted project settings and tool-protocol manifests executing on directory open, and an environment variable in a malicious repository redirecting API traffic before the trust prompt appeared.[58]

The generalization: `.mcp.json`, agent rule files, editor settings, and linter configuration are executable content. They are committed, they are rarely reviewed with the care given to source, and they are frequently excluded from code-owner requirements. Treat them as code, or better, as privileged code.

Local agent tooling is a lateral movement capability. In August 2025 the Nx build system's npm packages were compromised for approximately four hours through a workflow injection flaw. The post-install payload scanned for locally installed AI command-line tools, three of which were named in the published analysis, and invoked them with an authorization-framing prompt instructing them to act as "an authorized penetration testing agent" and enumerate the filesystem for interesting files.[59,60] Measured impact: 1,346 public repositories created for exfiltration, 2,349 distinct secrets leaked of which more than 1,100 remained valid at analysis time, followed by a second phase in which valid tokens were used to flip private repositories public, exposing 82,901 secrets across 2,399 repositories.[61]

This is the first well-documented case of malware using a developer's own AI agent as a privileged discovery tool, and it reframes what an agent installation is. It is not a productivity tool with a security footnote. It is a credentialed, filesystem-capable, network-capable process running under developer identity, and it should be inventoried, scoped, and monitored as one.

Dependency selection acquired a new failure mode. The definitive measurement of package hallucination analyzed 576,000 code samples across sixteen models and two languages, finding 19.7 percent of 2.23 million recommended packages hallucinated—205,474 unique non-existent package names.[62] The property that converts this from an error into an attack surface is repeatability: 43 percent of hallucinated packages recurred in all ten re-queries, and 58 percent recurred more than once in ten iterations. An attacker does not need to discover an organization's private namespace; the model supplies a public, reproducible target list, and the attacker registers the names and waits. A 2026 follow-up across five frontier models found rates compressed to between 4.6 and 6.1 percent—an order-of-magnitude narrowing of the inter-model spread, not a retirement of the threat—and identified 127 package names hallucinated identically by all five models, of which 53 remained registrable.[63] That follow-up is an unreviewed preprint by an independent researcher and should be cited as directional.

Version selection is a separate and arguably worse problem. A study across ten models and one thousand programming tasks found that 36.7 to 55.7 percent of tasks contained at least one known CVE in the dependency versions the model specified, that 62.8 to 74.5 percent of those CVEs were critical or high severity, and that 72.3 to 91.4 percent had been publicly disclosed *before the model's training cutoff*.[64] The models had the information and selected the vulnerable version anyway. This is systemic bias across all tested models rather than isolated error, and it means that dependency version pinning cannot be delegated.

The registry ecosystem is under sustained, automated attack. The Shai-Hulud npm worm's second wave, identified in November 2025, compromised 796 unique packages representing more than twenty million weekly downloads, moved from post-install to *pre-install* execution to eliminate human interaction and evade static scanners, harvested cloud workload credentials via instance metadata services, established command and control through self-hosted CI runners, and added a destructive fallback that attempted to destroy the victim's home directory when exfiltration failed.[65,66] More than five hundred unique users had credentials exfiltrated. Registry operators responded—npm permanently revoked all classic tokens in December 2025, moving to session-based authentication with two-factor enforcement at publish and trusted publishing via OIDC—which is meaningful hardening and does not change the posture a consuming organization should take.[67,68]

Review is saturated. Section 3.3 established the numbers. The practical consequence at this stage is that "a human reviews everything" is a policy that is already failing in organizations that believe they have it, and the failure is silent because the merge record looks identical either way.

13.2 Elements to Adopt

- **Agent identity distinct from human identity.** Every agent operating at A2 or above requires its own principal. Not a shared bot account, not a developer's personal access token, not a service account inherited from CI. The reasons are ordinary: attribution, revocation, scoping, review. The failure is ordinary too, and it is documented: the OWASP Non-Human Identities Top 10 leads with improper offboarding, and includes overprivileged identities, long-lived secrets, and identity reuse, all of which describe how agents are typically provisioned in practice.[69]

- **Ephemeral, task-scoped credentials.** No standing credentials in an agent execution context. Where the toolchain supports OIDC-based credential exchange, use it; where it does not, issue short-lived tokens bound to the task. The relevant test is a demonstration rather than a policy: compromise a test agent's context and enumerate what is obtainable and for how long it remains valid.
- **Execution isolation with default-deny egress.** Agents execute in non-privileged, isolated runtimes with no host filesystem or host network access, and outbound connectivity restricted to an enumerated allowlist. Package installation flows through an internal mirror treated as a production security boundary. Every permitted egress path has a named owner and a patch cadence.
- **Configuration files as privileged code.** Tool-protocol manifests, agent instruction and rule files, editor settings, and CI configuration require code-owner review, are covered by branch protection, and are scanned for concealed content—zero-width characters, bidirectional overrides, and other invisible-rendering techniques. Approval of a tool-protocol server binds to the configuration contents, not to the server's name.
- **Dependency governance that does not trust generation.** New dependencies introduced by agent-authored change require explicit approval against a maintained allowlist. Every dependency is version-pinned with an integrity hash. Package existence, age, download history, and maintainer provenance are checked before install rather than after. Internal namespaces are reserved on public registries to close the confusion path. Given the measured version-selection bias, dependency versions in agent-authored change are resolved by the toolchain against policy, not accepted as the model specified them.[64]
- **Provenance emission at authorship.** Agent-authored commits carry attribution in the commit record, are cryptographically signed, and link to a session record that identifies the model version, the invoking human, the tool set, and the task. The reference implementation observable in shipping platforms attributes the commit to the agent with the invoking human as co-author, signs it, and links each commit to session logs.[70] Where contributions may flow upstream, the disclosure convention that projects have converged on is a dedicated commit trailer, with the rule that agents never add Developer Certificate of Origin sign-off because only a human can certify it.[43]
- **A merge boundary that holds.** The most concrete published control set for a repository-resident coding agent, and one worth adopting whatever product implements it, comprises: only users with write access may invoke the agent; the agent pushes to exactly one designated branch; the agent cannot execute version control commands directly; its output opens as a draft pull request that it cannot mark ready; **it cannot approve its own pull requests**, so review requirements survive; CI workflows do not execute until a human with write access approves them; network access is firewall-restricted; and hidden characters are filtered from input.[71] Each of these is a separate control and each closes a specific path. The self-approval prohibition and the workflow approval requirement are the two that most directly preserve the separation of duties that DORA's technical standards require and that SOC 2 CC8.1 assumes.[4,3]
- **Review triage rather than uniform review.** Uniform review does not survive contact with the volume. Risk-tier the change instead: security-relevant paths, authentication and authorization code, configuration files, dependency manifests, infrastructure definitions, and public interfaces receive mandatory senior review regardless of size, while low-risk mechanical change receives lighter treatment. Automated review assists triage; it does not replace the human decision on the high-risk tier. And the practice worth borrowing from source-provenance standards is two-person review on protected branches, which is precisely what the SLSA Source Track's highest level requires.[72]

- **Measured review integrity.** An approval control with a very high approval rate and a very short median review time is not a control. Instrument approval rate, median and 95th-percentile review duration, and diff size at approval. Where those metrics degrade past a threshold, the correct response is to lower the merge rate, not to note the trend.
- **Concurrency governance.** Given a measured 41.7 percent cross-product conflict rate when agents from different vendors work the same repository concurrently, limit concurrent agent work streams per repository, partition agent scope by module or path, and require rebase-and-reverify rather than merge-and-hope.[32]

13.3 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
DEV-1	**Unique identity per agent deployment**	Distinct principal, no shared accounts	Lifecycle automated; deprovisioning on owner departure or inactivity; NHI review cycle separate from human
DEV-2	**No standing credentials in agent execution context**	Vaulted, scoped, rotated	Ephemeral task-scoped issuance; verified by live compromise test with measured validity window
DEV-3	**Isolated runtime with default-deny egress**	Containerized, non-privileged, allowlisted egress	Egress paths inventoried with owners and patch cadence; ephemeral runtimes destroyed per task
DEV-4	**Configuration files treated as privileged code**	Code-owner review required on agent and tool configuration	Concealed-character scanning; tool approval bound to content hash, not server name
DEV-5	**Dependency introduction governed**	Allowlist approval for new dependencies; pinning with integrity hashes	Automated existence, age, and provenance checks pre-install; internal namespaces reserved publicly
DEV-6	**Dependency versions resolved by policy, not by model output**	Manual verification of agent-specified versions	Toolchain resolves versions against vulnerability policy; model-specified versions never accepted directly
DEV-7	**Agent-authored commits carry signed, linked provenance**	Attribution present in commit record	Signed commits with session linkage; unsigned or unattributed agent commits rejected at push

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
DEV-8	**Agent cannot approve its own change**	Policy stated and configured	Enforced by platform ruleset; verified by test; bypass actors reviewed quarterly
DEV-9	**CI execution requires human authorization for agent-initiated change**	Configured for agent-authored pull requests	Enforced platform-wide; workflow approval logged and attributable
DEV-10	**Review risk-tiered with mandatory senior review on high-risk paths**	Path-based review requirements configured	Enforced by code owners; bypass requires documented exception with expiry
DEV-11	**Review integrity measured**	Approval rate and review duration reported	Thresholds trigger automatic merge-rate reduction rather than notification
DEV-12	**Secret scanning and static analysis blocking at merge**	Scanning enabled with alerting	Blocking gates with defined exception path; findings triage capacity measured against volume
DEV-13	**Concurrent agent work streams bounded per repository**	Limit set and observed	Scope partitioned by path; conflict rate monitored; cross-product concurrency restricted

13.4 Evidence to Request

- A live demonstration of credential exposure: compromise a test agent execution context and enumerate every credential obtainable and its remaining validity.
- The platform ruleset configuration showing the self-approval prohibition and the workflow approval requirement, plus the list of configured bypass actors and when each was last reviewed.
- Approval-rate and review-duration telemetry for the trailing quarter, broken out by whether the change was agent-authored.
- A random agent-authored commit traced to its session record, model version, invoking human, and reviewer.
- The dependency allowlist with its approval history, and evidence that a hallucinated or vulnerable package was actually blocked.
- The egress allowlist with owners and last-patched dates.

13.5 Failure Modes

- **Agents operating under human credentials.** The most common provisioning outcome and the one that makes everything downstream unanswerable. Attribution fails, revocation is impossible without disabling a person, and access review reports a human population that is partly machine.

- **Configuration excluded from review.** The `.github` directory, the agent rule files, the tool manifests—reviewed casually or not at all, because they are not "code." Every major disclosed vulnerability in this class went through exactly this gap.
- **Self-approval available and unnoticed.** Frequently a default rather than a decision. Where an agent can approve its own pull request, the separation-of-duties requirement in DORA's technical standards, the PCI DSS requirement for review by someone other than the originating author, and the SOC 2 change-authorization criterion are all unsatisfied simultaneously, and no control reports it.
- **Bypass actors accumulating.** Rulesets grow exceptions. An agent added as a bypass actor to unblock a delivery deadline remains one indefinitely, and the ruleset that appears to enforce review does not enforce it for the principal generating the most change.
- **Security findings outrunning triage.** Where generated volume produces findings faster than they can be adjudicated, the observable response is suppression—thresholds raised, rules disabled, findings bulk-dismissed. The scale of the problem is acknowledged at the ecosystem level: the Linux Foundation announced $12.5 million in grant funding in March 2026, contributed by seven major technology and model-provider organizations and managed through Alpha-Omega and the Open Source Security Foundation, citing "an unprecedented influx of security findings, many of which are generated by automated systems" and committing funds to help maintainers with "the triage and processing of the increased AI-generated security reports they are currently receiving."[73] Enterprises face the same arithmetic internally, and should measure findings-generated against findings-adjudicated as a first-class metric.
- **Local agent installations outside inventory.** A developer installs a capable agent, points it at a repository, and grants it credentials, entirely outside procurement and entirely outside the controls above. This is not misconduct; it is the path of least resistance, and it is closed by making the governed path easier rather than by policy.

14 Stage Five — Testing and Verification

OBJECTIVE

Establish that the system meets its requirements—where a substantial fraction of the code was generated, some components are probabilistic, and the volume of change exceeds what conventional verification was sized for.

14.1 What Changes at This Stage

Three distinct verification problems now coexist, and they require different machinery.

Verifying generated code is the familiar problem at unfamiliar volume, with a shifted defect distribution. The measured 55 percent security pass rate and the shift toward architectural and privilege-boundary flaws mean that verification effort should be reallocated: away from the syntactic and local defects that generation has largely eliminated, toward the classes generation systematically produces.[15,36]

Verifying probabilistic components is a genuinely new problem for most engineering organizations. There is no input for which output is guaranteed, so acceptance cannot be binary. The discipline that applies is statistical, and the reference treatment argues that evaluation should be understood as sampling from an unobserved super-population, which makes confidence intervals meaningful rather than decorative and makes power analysis a precondition for sizing an evaluation set.[74] A two-point difference on a two-hundred-item evaluation set is usually indistinguishable from noise, and most production evaluation suites report raw pass rates without intervals.

Verifying the agents themselves is the least developed area in this entire framework. There is no published enterprise standard for agent qualification, no accepted protocol, and no agreed criterion for when an agent may be granted repository write access. Available benchmarks measure capability on curated tasks, and benchmark saturation is now actively misleading: leading systems score above 95 percent on the widely used verified software engineering benchmark, with seven of eighty-six evaluated models at or above that mark, while real-world agentic pull request merge rates sit at 63.1 percent and the security pass rate for generated code sits at 55 percent.[75,31,15] Benchmark performance is not a qualification signal. Say so internally before someone cites it in a procurement decision.

A fourth shift cuts across all three: **the test suite is now also generated**. When the same model writes the implementation and the tests, passing tests establish self-consistency rather than correctness. This is the specific case where the independence principle in Section 1.3 binds hardest.

14.2 Elements to Adopt

Independence between generation and verification. Tests for agent-authored code are derived from the requirement, not from the implementation, and preferably not by the same agent in the same session. Where an agent generates both, an independent check (mutation testing, property-based testing, coverage of specified

behaviors rather than of lines, or human review of the test's intent) establishes that the tests would fail if the implementation were wrong. Mutation testing is the most practical mechanical answer available, because it directly measures whether a test suite detects injected defects.

Evaluation harnesses as versioned, owned artifacts. For every model-backed capability, maintain an evaluation set, a metric definition, a threshold, and a measurement cadence. Own the metric definitions rather than inheriting them: "faithfulness" computes differently in each major evaluation framework, and scores are not portable between tools.[76] The transport layer is converging faster than the semantics—a point Section 16 returns to—but evaluation definitions remain an organizational artifact.

Time-windowed evaluation sets. Any evaluation corpus that has existed on the public internet for a year should be assumed present in the next model's training data. The reference design pattern annotates problems with release dates so a model can be scored only on material published after its training cutoff.[77] The transferable enterprise practice is a rolling holdout of recent, never-published production cases, plus periodic re-derivation of the golden set from live traffic. This is also the honest answer to why evaluation scores sometimes improve without the system improving.

Validated judges, or no judges. Model-as-judge evaluation is widely deployed and widely over-trusted. A systematic evaluation of twenty-one judge models across three benchmarks, comprising roughly 541,000 individual judgments, found a 33 to 41 percentage point gap between exact-match agreement and chance-corrected agreement, meaning headline agreement figures systematically overstate judge quality; found judge rankings shifting by up to fourteen positions depending on which benchmark was used; and—the most instructive finding—found two production-deployed judges with test-retest reliability above 0.95 while exhibiting position bias above 0.10.[78] They were consistently wrong in the same direction, which is the worst failure mode precisely because it presents as stability. A judge that gates a release must itself be validated against human labels, and reliability must not be mistaken for validity.

Deterministic criteria wherever they are available. Not every acceptance criterion for a model-backed feature requires a judge. Schema conformance, required-field presence, refusal behavior, tool-call validity, latency, cost per task, and prohibited-content checks are all deterministic and should carry as much of the acceptance load as possible, with judged criteria reserved for what genuinely cannot be checked mechanically.

Security testing reallocated to the observed defect profile. Given measured pass rates of 15 percent on cross-site scripting and 13 percent on log injection, and a defect shift toward privilege escalation and architectural flaws, verification should weight output-encoding contexts, authorization boundaries, and inter-component trust relationships far more heavily than the syntactic classes.[15,36] Static analysis remains necessary and is weakest exactly where the profile is shifting, which argues for supplementing it with architectural review and authorization-path testing rather than for buying more scanners.

Adversarial testing of the delivery pipeline itself. Red team the toolchain, not only the product. Concretely: attempt prompt injection through an issue, a pull request description, a code comment, a dependency README, and a tool response, and establish whether an agent's behavior changes. Attempt to introduce a dependency that does not exist. Attempt to modify an agent rule file with concealed characters and see whether review catches it. Attempt to have an agent approve its own change. The published red-teaming methodology worth borrowing from is scored

as an attack success rate across a defined strategy set, which produces a comparable number rather than a narrative.[79] The relevant risk categories for agents specifically are prohibited actions, sensitive data leakage, and task adherence.

Agent qualification before write access. In the absence of a standard, adopt a defensible internal protocol and be explicit that it is internal. The practical sequence is: sandboxed trial in an isolated environment; shadow mode, where the agent produces diffs that are scored against what humans actually did but never merged; canary scope on a low-blast-radius repository with full review; and then graduated scope expansion with measured outcomes. The qualification criteria should be organizational—defect rate, review rework rate, scope adherence, conflict rate—rather than benchmark scores.

Non-functional verification restored to the gate. Accessibility conformance, performance budgets, and resource ceilings become pipeline gates rather than periodic audits. The population-scale accessibility regression is the strongest available evidence for what happens when they are not.[37]

14.3 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
VER-1	**Tests for agent-authored code independent of the generating context**	Tests derived from requirements; independence stated in policy	Mutation or property-based testing verifies suite sensitivity; suites failing sensitivity thresholds block merge
VER-2	**Evaluation harness exists for every model-backed capability**	Corpus, metric, and threshold defined and owned	Evaluation runs as a merge and release gate; threshold breach blocks promotion
VER-3	**Statistical acceptance criteria with intervals**	Thresholds documented	Confidence intervals reported; evaluation sets power-analyzed for the effect size being detected
VER-4	**Evaluation corpora refreshed and contamination-resistant**	Scheduled refresh with a named owner	Rolling holdout of never-published production cases; contamination checked on model version change
VER-5	**Judge models validated against human labels before gating**	Validation performed and recorded	Periodic revalidation; bias and chance-corrected agreement reported alongside raw agreement
VER-6	**Deterministic checks carry maximum feasible acceptance load**	Deterministic criteria specified where available	Judged criteria justified individually; ratio of deterministic to judged criteria reported

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
VER-7	**Security verification weighted to the observed defect profile**	High-failure classes explicitly tested	Authorization-path and output-encoding testing mandatory on affected changes
VER-8	**Delivery pipeline adversarially tested**	Annual toolchain red team	Scheduled, scored as attack success rate, covering injection, dependency, configuration, and approval paths
VER-9	**Agents qualified before write access**	Documented qualification with named approver	Staged qualification—sandbox, shadow, canary—with organizational outcome criteria and re-qualification on version change
VER-10	**Non-functional requirements verified at the gate**	Accessibility and performance checks in pipeline	Blocking gates with budgets defined per requirement; regressions block release
VER-11	**Findings triage capacity measured against findings volume**	Ratio reported	Ratio governs merge rate; sustained deficit triggers autonomy tier reduction

14.4 Evidence to Request

- The evaluation set for a model-backed capability, with the date it was last refreshed and the method by which contamination is prevented.
- The judge validation record: agreement with human labels, chance-corrected, with bias measurements.
- Mutation testing results for a representative agent-authored module—a high line-coverage figure alongside a low mutation score is the specific finding this control exists to surface.
- The most recent toolchain red team report and its attack success rate.
- The agent qualification record for the agent with the broadest current write access, including who approved it and against what criteria.
- Findings generated against findings adjudicated for the trailing quarter.

14.5 Failure Modes

- **Self-consistent test suites.** The implementation and the tests share a misunderstanding, the suite passes, and coverage metrics look excellent. This is not detectable by any coverage measure and is the single most likely way a defect reaches production in a heavily generated codebase.

- **Evaluation theater.** An evaluation set exists, has not been updated in a year, has probably leaked into training data, and reports a number that goes up. Nobody has computed an interval. The number is reported to executives as a quality metric.
- **Judges gating releases unvalidated.** A model scores another model's output, the score is treated as ground truth, and the systematic bias in the judge is invisible because the judge is consistent.
- **Benchmark scores used as qualification.** A near-saturated public benchmark is cited to justify granting an agent broader write access. The benchmark measures capability on curated tasks under no organizational constraint, and the merge-rate and security data say it does not transfer.
- **Triage suppression.** The finding volume exceeds capacity, so the threshold rises. The dashboard improves. This is the failure mode most likely to be presented internally as a success.

15 Stage Six — Deployment and Release

OBJECTIVE

Move change into production with an auditable record of what shipped, who authorized it, and what produced it—and with rollback paths that work when the failure is a behavior change rather than an error.

15.1 What Changes at This Stage

The release record must answer a question it was not designed to answer. Historically, "who wrote this" and "who approved this" were both answerable from the commit and review record, and both answers were people. The release record now needs to distinguish agent-authored from human-authored change, identify the model version and the authorizing human, and do so in a form that survives an audit eighteen months later. No SBOM format has a native construct for "this source file was generated by an AI agent"; both major formats describe models and datasets reasonably well and neither describes generated code provenance.[80,81] The gap is real and should be closed by convention plus attestation rather than waited out.

Rollback triggers do not fire on the failure that matters. In conventional progressive delivery the trigger is an error rate: a 5xx, a latency spike, a crash. A model-backed feature that degrades usually produces no error at all. The service returns 200 with a well-formed response that is now subtly worse. The rollback trigger must therefore be a distribution comparison against a baseline—evaluation score, judge score, refusal rate, tool-call error rate, task completion rate—rather than an error threshold. There is no published enterprise standard for this and no independent research on it; it is named here as an open problem in Section 23 and as a design requirement regardless.

Separation of duties is the regulatory pressure point. Article 17 of the DORA regulatory technical standards requires independence between the function approving a change and the functions requesting and implementing it.[4] PCI DSS 6.2.3 requires pre-release code review, with 6.2.3.1 requiring a reviewer other than the originating author where the review is manual, and permitting automated review as an alternative.[2] SOC 2 CC8.1 requires authorization preceding implementation.[3] An agentic pipeline satisfies all three only if the agent cannot approve its own change and cannot authorize its own workflow execution—which is why those two controls appear in Section 13 and are re-tested here.

The honest framing for an audit conversation is this: these criteria do not break under agentic delivery, but the control relocates. It moves from the individual change to the policy, the enforced ruleset, and the evidence pipeline governing the agent. That is an interpretive position, and it should be presented to auditors as one rather than as a settled reading.

Release governance now has a shipped reference implementation, and feature availability is not maturity. Platform capability in this area advanced substantially through 2025 and 2026—enterprise control planes for defining approved models, restricting which agents teams may use, managing agent identity, and configuring audit logging;

source-controlled agent instruction files; and organization-wide code quality visibility.[52] These are real capabilities. Whether an organization has configured them is a separate question from whether its vendor ships them, and the two are routinely conflated in maturity self-assessments.

15.2 Elements to Adopt

Provenance in the release record. Every release carries a manifest identifying agent-authored components, the model versions involved, the authorizing humans, and the verification that ran. Where the toolchain supports build provenance attestation, emit it; the applicable standard defines a build track at levels 0 through 3 and, since November 2025, a source track at levels 1 through 4, where the highest source level requires "two trusted persons to review all changes to protected branches."[72] That source-track requirement is currently the strongest available control for agent-authored code precisely because it addresses the problem through review rather than through provenance metadata that does not yet exist.

A bill of materials that includes the AI dependencies. The general SBOM baseline was reissued in July 2026 with ten new fields including author signature, component hash and hash algorithm, component license, and generation context.[82] Separately, a government-backed AI SBOM specification defines seven clusters (metadata, system-level properties, models, dataset properties, infrastructure, security properties, and key performance indicators) requiring documentation of model lineage and dataset provenance.[83] Both are voluntary. Adopt them anyway: the elements are agreed even though interoperability is not, and being early on this is cheap relative to being late.

Signed artifacts and verified provenance at the release gate. Keyless signing infrastructure has matured to the point where this is an ordinary engineering task rather than a program: short-lived certificates bound to a workload identity, with the signing event recorded in an append-only transparency log.[84] Model artifacts have a corresponding signing specification, PKI-agnostic and layered on the same envelope and statement formats, with contributions from most major infrastructure vendors.[85]

Deployment authority separated from generation authority. An agent that writes code does not deploy it. An agent that deploys does not write. Where an organization wants both, they are two deployments with two identities, two tiers, and two owners. This is the cheapest structural control in the framework and it survives every subsequent argument about tooling.

Progressive delivery keyed to quality distributions. Canary and shadow deployment are mature practices and nothing about them changes mechanically. What changes is the promotion criterion: shift traffic on evaluation score distribution, refusal rate, and task completion rate rather than on error rate alone, and define the rollback trigger before the deployment rather than during the incident. Runtime configuration of prompts, instructions, and model selection outside the application binary is the capability class that makes this practical, since it allows a model or prompt change without a redeployment and a revert without a rebuild.[86]

Model version changes treated as releases. A provider model change is a production change to your system, whether or not your code changed. It goes through the release gate, runs the evaluation suite, and is announced to whoever operates the service. A pinned dated snapshot is the baseline; aliases are prohibited in production paths.

Operational readiness for agent-touched systems. Before release, confirm that the system emits the telemetry Section 16 requires, that a kill switch exists for any agent operating against it with a measured time-to-effect, that on-call staff can distinguish a model failure from a code failure, and that the runbook covers the case where the agent's own account of what happened is wrong. That last item is not hypothetical: in the Replit incident the agent reported that rollback was impossible when it was not.[14]

15.3 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
REL-1	**Release record distinguishes agent-authored from human-authored change**	Manifest produced per release	Provenance attestation emitted and verified; releases without provenance blocked
REL-2	**SBOM produced covering software and AI components**	SBOM generated per release	AI components documented including model lineage and dataset provenance; SBOM verified at consumption
REL-3	**Artifacts signed with verifiable provenance**	Signing in place for release artifacts	Signature and provenance verified as a release gate; unverified artifacts cannot promote
REL-4	**Separation of duties preserved across generation, approval, and deployment**	Configured and documented	Enforced by platform; tested quarterly by attempting self-approval and self-deployment
REL-5	**Deployment authority distinct from authoring authority**	Separate identities for authoring and deploying agents	Enforced by credential scope; cross-use blocked and alerted
REL-6	**Rollback triggers defined on quality distribution, not error rate alone**	Quality-based triggers defined for model-backed features	Automated rollback on distribution shift with measured time-to-effect
REL-7	**Model version changes gated as releases**	Change control applied to model version changes	Evaluation suite gates version promotion; alias use blocked in production paths
REL-8	**Operational readiness verified for agent-touched systems**	Readiness checklist completed	Kill switch tested with measured time-to-effect; telemetry coverage verified before promotion

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
REL-9	**Release approval evidence retained and attributable**	Approval records retained	Tamper-evident retention; approvals attributable to a named human for every agent-authored change

15.4 Evidence to Request

- A release manifest showing the agent-authored share and the model versions involved.
- A signature and provenance verification log for a recent release.
- The result of a quarterly separation-of-duties test—an actual attempt to have an agent approve or deploy its own change, and the record of it failing.
- The rollback trigger definition for a model-backed feature, and the last time it fired or was exercised.
- The SBOM for a shipped release, checked for whether AI components appear at all.

15.5 Failure Modes

- **The release record that cannot answer the authorship question.** Discovered during an incident, an audit, or a legal inquiry, at which point it is not fixable retroactively.
- **Rollback that handles unavailability and not degradation.** The fallback path fires when the provider is down and never fires when the model is confidently wrong, which is the far more common and far more damaging failure.
- **Separation of duties assumed rather than tested.** Configured once, never verified, and quietly broken by a ruleset exception added eight months later to unblock a release.
- **Alias drift in production.** A floating model alias somewhere in the configuration means the provider changes the system's behavior on their schedule. It is usually found because behavior changed, not because anyone audited for it.
- **Feature availability recorded as maturity.** The platform supports an agent control plane; nobody configured it; the maturity self-assessment claims it. Independent assurance exists to catch exactly this.

16 Stage Seven — Operations and Maintenance

OBJECTIVE

Operate, monitor, and maintain a system whose code is substantially generated, whose behavior may be probabilistic, whose cost is consumption-driven, and whose long-term comprehension by humans cannot be assumed.

16.1 What Changes at This Stage

Operations inherits every deferred decision from the preceding six stages, and it inherits them at a scale that was set elsewhere. Three shifts matter most.

Validation stops being a gate and becomes a process. This is the one place where a published international standard has already made the argument. ISO/IEC 5338 classifies AI system life cycle processes as generic, modified, or AI-specific relative to the underlying software and systems lifecycle standards, and identifies exactly three AI-specific processes: knowledge acquisition, AI data engineering, and **continuous validation**.[6] Continuous validation formalizes the position that validation is not something a system passes once but something that runs for its operational life, because behavior drifts. For an agentic SDLC framework this is the standards-body precedent for the claim that the release boundary is no longer where verification ends. Cite it; it is more durable than any argument this document could construct.

Four distinct phenomena get collapsed into the word "drift," and they need different monitoring. *Model drift* is provider-side: the model changes beneath a stable API surface. The canonical demonstration measured one model's accuracy on a specific task falling from 84 percent to 51 percent across three months, alongside reduced instruction-following and increased code-formatting errors.[87] That paper drew methodological criticism at the time, and the criticism is fair on the specific task, but the core claim—that a production model service's behavior can shift materially within months without a version change visible to the caller—has not been refuted. *Data drift* is input-side and is the classical machine learning problem, with the classical treatment still applying.[88] *Prompt drift* is accumulated uncoordinated edits to prompts and system instructions, often across teams, usually with no test coverage; it is real, widely observed, and has essentially no research literature, so the mitigation—version prompts as code, require an evaluation diff on change, treat a prompt edit as a deployment—is sound engineering reasoning rather than an evidenced practice, and should be presented internally as such. *Evaluation set staleness* is covered in Section 14.

Correctness becomes unobservable at request time. Classical service level objectives assume a binary per-request outcome. For an agent, you frequently cannot tell whether a response was correct without a downstream signal or a human. That breaks the error budget model at its foundation, because a budget cannot be debited for failures that are not detected. The workable partial answer is to define indicators on observable proxies and to run correctness as a sampled offline evaluation feeding a separate quality metric. The best-specified public proposal defines seven indicator types—task success rate, tool call accuracy, 95th-percentile latency, cost per task, policy compliance, scope chain depth, and hallucination rate—with conventional burn-rate alerting, a cost-guard kill switch triggering at

95 percent of budget consumption, and a per-agent circuit breaker.[89] Two of those indicators are genuinely novel and worth adopting whatever the implementation: *scope chain depth*, measuring how far an agent has traversed from its authorized boundary, and *cost per task* treated as a reliability signal rather than a finance metric. That work is published open source in public preview, with no production efficacy data, and should be treated as a well-specified proposal rather than established practice.

16.2 Elements to Adopt

Telemetry that covers the agentic layer. The transport is settled and the semantics are not, and being precise about that distinction saves an architecture argument. The OpenTelemetry generative AI semantic conventions have moved to a dedicated repository, and every span, event, metric, and `gen_ai.*` attribute is marked Development status with no tagged release—the only stable attributes appearing in these signals are inherited core ones.[90] Anyone claiming the conventions are stable is wrong. They are nonetheless the right thing to emit, with attribute names treated as a versioned contract with your own pipeline. Two design decisions are worth knowing. The conventions define a distinct planning operation alongside agent invocation and tool execution, which makes an agent's task decomposition separately observable from what it then does. And an evaluation result event carries an evaluation name and a score value, which is the first credible standard bridge between the evaluation layer and the observability layer. Message content attributes are opt-in by default for privacy reasons and should be enabled under a data classification policy rather than globally.

Per-agent, per-task cost observability. Cost is a reliability and security signal, not only a finance one. Runaway consumption is the earliest observable indicator of an agent in a loop, an agent being manipulated, or an agent operating outside its intended scope. The unit metric is cost per task, not cost per token, precisely because agentic workloads consume far more tokens per unit of work than conversational use and unit-price declines therefore do not imply bill declines. The relevant billing standard currently handles AI through existing columns—consumed quantity and unit carrying token counts, SKU identifiers marking token charges—rather than through AI-native fields, which is workable and worth knowing is a stopgap.[91]

Kill switches that are tested and measured. For every agent operating against production, a kill switch that halts the individual agent, the agent class, and the whole agent estate, each with a documented and measured time-to-effect. Untested kill switches are a documentation artifact. The specific gap to check for is agents running inside SaaS platforms, third-party tools, and developer environments, where the orchestrator's kill switch does not reach.

Incident response that accounts for unreliable self-report. Add three items to the runbook. First, a discriminator between model failure and code failure: re-run the same input against the pinned model and the deployed code, where reproducible failure implicates code or prompt, non-reproducible failure implicates model non-determinism or provider-side change, and failure that reproduces against a new snapshot but not the old one implicates a provider change. This is sound reasoning rather than published practice and should be labeled as such. Second, evidence preservation from ephemeral agent runtimes, including snapshot-on-anomaly before a container is destroyed, because the ephemerality adopted as a security control destroys the forensic record. Third, explicit guidance that an agent's account of its own actions is evidence to be verified rather than testimony to be believed.

Maintenance practice that addresses comprehension. The maintainability data in Section 3.6 and the comprehension finding in Section 2 point at the same operational risk from different directions: systems accumulating code that nobody fully understands, with debt persisting at measured rates and security issues

persisting nearly twice as long as other categories.[39] Practical responses: require an explanatory record for non-trivial agent-authored change at the time of authorship rather than at the time of confusion; schedule and fund consolidation work explicitly, since the data says it will not happen spontaneously; measure duplication and reuse as operational metrics rather than aesthetic ones; and maintain named human ownership of every subsystem with a stated expectation that the owner can explain it.

Non-human identity lifecycle discipline. Agent identities are provisioned quickly and deprovisioned rarely. Bring them into access review at the same cadence as privileged human accounts or more frequently, given their number and rate of change, and report non-human and human identity governance separately. A combined figure conceals a much worse machine posture behind good human hygiene, and the improper-offboarding failure that leads the published non-human identity risk list is exactly what a combined figure hides.[69]

Continuous validation in production. Run the evaluation suite against production traffic on a schedule, not only at release. Alert on distribution shift rather than on errors. This is the operational implementation of the ISO/IEC 5338 process and the only reliable detector of provider-side model drift.

16.3 Controls

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
OPS-1	**Agent and model telemetry centrally collected**	Spans, tool calls, outcomes, and token usage collected	Planning and execution spans distinguished; evaluation results emitted as telemetry; coverage measured across the estate
OPS-2	**Cost observable per agent and per task with enforced ceilings**	Cost attributed to agent and team	Per-task unit cost tracked; anomaly detection with automated halt at a defined budget threshold
OPS-3	**Continuous validation running against production**	Scheduled evaluation against production traffic	Automated alerting on distribution shift; shift triggers defined response including rollback
OPS-4	**Model version drift detected**	Provider notices monitored; snapshots pinned	Canary evaluation detects behavioral change within a pinned snapshot; deprecation calendar maintained with owners
OPS-5	**Prompt and instruction changes version-controlled and evaluation-gated**	Prompts in version control	Prompt change requires evaluation diff and follows the release path
OPS-6	**Kill switch tested per agent, agent class, and estate**	Kill switch exists and is documented	Tested with measured time-to-effect, including agents in third-party platforms and developer environments

ID	CONTROL	MINIMUM BAR (L2)	ENFORCED STATE (L3)
OPS-7	**Evidence preserved from ephemeral agent runtimes**	Retention policy defined	Automatic snapshot-on-anomaly before runtime destruction; preservation tested
OPS-8	**Incident response covers model-versus-code discrimination**	Runbook includes the discriminator	Exercised in rehearsal; agent self-report treated as evidence requiring verification
OPS-9	**Agent identities in access review, reported separately from human**	Included in review cycle	Automated deprovisioning on inactivity, owner departure, or project closure; separate NHI reporting to executives
OPS-10	**Maintainability and comprehension monitored**	Duplication and reuse metrics reported	Thresholds trigger funded consolidation work; every subsystem has a named owner who can explain it
OPS-11	**Agent reliability indicators defined and monitored**	Task success and tool-call error rate tracked	Full indicator set including scope chain depth; circuit breakers halt rather than alert

16.4 Evidence to Request

- Kill switch test results with measured time-to-effect, by agent class, including at least one agent operating inside a third-party platform.
- Telemetry coverage: what fraction of the agent estate emits action telemetry, and what fraction of what is emitted is actually analyzed.
- The most recent continuous validation run against production, with its distribution comparison.
- A cost anomaly and the response it triggered.
- The non-human identity access review, reported separately from human identities.
- Duplication and reuse trend over the last four quarters, with any consolidation work that was actually funded.

16.5 Failure Modes

- **Recording mistaken for detection.** Comprehensive agent telemetry that nobody and nothing analyzes is an evidentiary asset, not a control. Instrument the generated-to-analyzed ratio explicitly, because it degrades silently.
- **Cost visible only in aggregate.** A monthly bill with no per-agent or per-task attribution cannot detect a looping agent, cannot support a chargeback conversation, and cannot inform a tier decision.

- **Drift monitored only at version boundaries.** The organization watches for announced model retirements and does not watch for behavioral change within a pinned snapshot, which is the drift that actually surprises people.
- **Kill switches that do not reach.** They exist in the orchestrator and not for the agent embedded in a SaaS platform, the agent in a partner's environment, or the agent running on a developer's laptop against corporate credentials.
- **Comprehension debt as a permanent unbudgeted liability.** It produces no ticket and no alert. It surfaces as an inability to change something, usually urgently, usually during an incident, and by then the people who could have explained it have moved on.

PART IV

The Constraint Layer

What determines how far the operating model can be pushed. These are not the thesis; they are what makes velocity survivable.

17 Cross-Cutting Domains

Four concerns refuse to localize to a stage. They are treated here at full depth rather than repeated seven times, and the crosswalk in 12.5 shows where each binds.

17.1 X1 Agent Identity and Authorization

OBJECTIVE

Every AI participant in delivery is a distinct, governable principal whose authority is bounded, inspectable, and revocable.

This is the foundation the rest of the framework stands on, and it is the one most often skipped because agents work perfectly well without it. Attribution, revocation, scoping, review, and incident scoping all become tractable when agents have their own identities and all become guesswork when they do not.

WHAT LEVEL 2 REQUIRES

One identity per agent deployment, with no sharing across deployments and no agent operating under a human's credentials. A documented lifecycle covering registration, approval, issuance, review, and deprovisioning. Inclusion in access review at the cadence applied to privileged human accounts. A registry entry naming the accountable human, the autonomy tier, the permitted scope, the tool set, the environments, and the decommission criteria.

WHAT LEVEL 3 ADDS

Task-scoped delegation, where the agent acts with authority derived from and bounded by the invoking principal rather than with standing authority of its own. Inspectable delegation chains, so that when one agent invokes another the authority passed is recorded and constrained and the second never inherits more than the first held. Ephemeral credentials issued at point of use with lifetimes in minutes. Automated deprovisioning tied to owner departure, project closure, or inactivity. Separation of the identity planes—the agent's identity for authentication, the invoking human's for authorization scope, the workload's for infrastructure access—never collapsed into a single credential.

STANDARDS POSITION

No published standard governs this. The NCCoE concept paper on software and AI agent identity and authorization is in comment review, and NIST's agent standards initiative names agent authentication and identity infrastructure as research pillars.[11,12] The usable interim reference is the OWASP Non-Human Identities Top 10, whose leading entries—improper offboarding, secret leakage, overprivileged identities, long-lived secrets, and identity reuse—describe the observed failure pattern precisely.[69]

EVIDENCE

Access review records covering agent identities, reported separately from human identities. A delegation chain trace for a representative multi-agent workflow—where the trace cannot be reconstructed, accountability is unresolvable regardless of what the policy says. Deprovisioning logs showing agents actually removed rather than marked inactive. Median credential lifetime across the agent estate.

17.2 X2 Provenance and the Attestation Chain

OBJECTIVE

Establish a verifiable record spanning design through operations of what was built, by whom or by what, under whose authority, and subject to which verification.

THE HONEST STARTING POSITION

There is no standard for cryptographically attesting that a given code artifact was AI-generated. What exists is a social convention in the form of commit trailers, which are unsigned and unverifiable; platform-specific co-authorship metadata, which has already demonstrated it can be wrong; and source provenance attestation that records how a revision was created without a field for what generated its content.[43,16,72] An enterprise framework should name this rather than paper over it.

WHAT CAN BE BUILT TODAY

The layers are individually mature even though the composition is not. Attestations follow a four-layer model of envelope, statement, predicate, and bundle, with a graduated open source project supplying the container format and a vetted predicate registry covering provenance, verification summaries, SBOMs, test results, and vulnerability data.[92] The build track supplies provenance at levels 0 through 3; the source track, added in November 2025, supplies revision provenance at levels 1 through 4 with two-person review at the top level.[72] Keyless signing with transparency logging is production-grade.[84] Model artifacts have their own signing specification layered on the same formats.[85] A graph-based query layer exists for making heterogeneous attestations answerable as questions rather than merely storable as files.[93]

THE COMPOSITION TO BUILD

Bind an organizational predicate carrying agent authorship metadata (agent identity, model and version, invoking human, session reference, autonomy tier, verification performed) into the attestation chain alongside build and source provenance. This is a custom predicate today. It is the right custom predicate to build, because the elements are stable even though the schema is not, and because a signed attestation makes claims checkable in a way that a commit trailer does not.

WHAT TO REQUIRE OF OTHERS

SBOM content per the reissued 2026 minimum elements, and AI component documentation per the G7-backed AI SBOM clusters where models are involved.[82,83] Signature verification at consumption, not merely at production.

EVIDENCE

A verification log showing an artifact rejected for missing or invalid provenance—a chain that has never rejected anything has not been tested. An end-to-end trace for one release, from requirement identifier through agent-authored commit, session record, verification results, signed artifact, and release manifest.

17.3 X3 Evaluation Infrastructure

OBJECTIVE

Maintain the organizational capability to determine whether a probabilistic component or an agent is behaving acceptably, independent of any vendor's tooling.

Evaluation appears in requirements as thresholds, in verification as gates, in release as promotion criteria, and in operations as continuous validation. Treating it as a per-stage activity produces four incompatible implementations. It is infrastructure.

WHAT LEVEL 2 REQUIRES

A registry of evaluation sets with owners, metric definitions, thresholds, and refresh cadence. Metric definitions owned by the organization rather than inherited from a tool, since the same metric name computes differently across frameworks and scores do not port.[76] Evaluation results retained as durable artifacts.

WHAT LEVEL 3 ADDS

Contamination-resistant corpus management with rolling holdouts of never-published production cases. Statistical acceptance with reported confidence intervals and evaluations sized by power analysis for the effect being detected.[74] Validated judges with chance-corrected agreement and bias reported alongside raw agreement.[78] Evaluation results emitted as telemetry so that gate results and production behavior are queryable together.[90] Evaluation as a merge and release gate rather than a report.

SELECTION POSTURE

Choose the harness for auditability and portability rather than for metric breadth. The relevant properties are local execution without data egress, declarative test definitions in version control, first-class CI integration, sandboxed execution of untrusted model output, and the ability to export raw results. Where an evaluation harness must be defensible to a regulator, provenance of the harness itself becomes a selection criterion.

EVIDENCE

The evaluation registry with refresh dates. A judge validation record. A merge blocked by an evaluation gate. The confidence interval reported alongside a threshold decision.

17.4 X4 Governance, Assurance, and Workforce

OBJECTIVE	Ensure autonomy expands through decisions with owners, that control claims are independently verified, and that the organization retains the human capability the framework depends on.
GOVERNANCE	A review body with authority to block deployment and to reduce autonomy tiers. Written policy covering agent registration, tier assignment, tier escalation, scope change, and decommissioning. A named accountable human for every A2-and-above deployment. Explicit governance of exceptional work (an agent given elevated credentials for a migration, extended tool access for an investigation, or reduced guardrails for a specific purpose), with a named approver, a time box, monitoring, and a defined termination condition. A review board that has never declined a request is a rubber stamp, and the absence of a single blocked or tier-reduced deployment in a year is itself the finding.
APPROVAL DESIGN THAT RESISTS FATIGUE	Where humans approve agent actions, instrument the approval. Batch where batching is safe, risk-weight escalation, and measure approval rate against review duration. An approval control with a very high approval rate and a very short median review time is documentation. This applies with particular force to the review integrity metric in Section 13, because that is where the volume lands.
INDEPENDENT ASSURANCE	Internal audit or an external party verifies maturity claims against producible evidence rather than self-assessment. This exists specifically to catch the failure mode where platform capability is recorded as organizational maturity.
WORKFORCE	Two obligations that are usually treated as culture and are actually risk controls. First, deliberate management of skill formation: the measured comprehension gap between delegation-style and interrogation-style usage is large, and an organization whose staffing model assumes junior engineers become senior engineers has a direct interest in which pattern it encourages.[18] Second, protection of the absorption capacity itself. Senior engineering attention is the binding constraint in agentic delivery; it is also the resource most easily consumed by review volume, and the one whose depletion is least visible until it is gone.
EVIDENCE	Review board minutes showing at least one blocked or tier-reduced deployment. Approval-rate and review-duration telemetry. The register of active exceptional-access grants with expiry dates. The most recent independent assurance report. Attrition and tenure data for senior engineers, read alongside review load.

17.5 Domain-to-Stage Crosswalk

DOMAIN	PLANNING	REQUIREMENTS	DESIGN	DEVELOPMENT	VERIFICATION	RELEASE	OPERATIONS
X1 **Identity and Authorization**	Tier proposed	Agent scope specified	Blast radius bounded	**Primary** — identity, credentials, isolation	Qualification before write access	Deployment authority separated	**Primary** — lifecycle, review, deprovisioning
X2 **Provenance and Attestation**	Obligations scoped	Provenance requirements captured	Attestation architecture designed	**Primary** — emission at authorship	Verification recorded as attestation	**Primary** — release manifest, SBOM, signing	Retention and forensic use
X3 **Evaluation Infrastructure**	Capability funded	**Primary** — thresholds and corpora defined	Designed for observability	Harness maintained as code	**Primary** — gates and validation	Promotion criteria	**Primary** — continuous validation
X4 **Governance and Workforce**	**Primary** — tier authority, absorption, cost	Requirements review authority	Architecture review authority	Review integrity, exception register	Assurance of verification claims	Release approval authority	**Primary** — assurance, NHI reporting, skill formation

18 The Control Maturity Model

Section 4 gives the organizational progression—Assisted, Delegated, Owned—and names the five control maturity levels the sections since have been written against. This section scores them: the full grid, and the control maturity that gates the progression. They are different axes, and the relationship between them is the framework's central operating rule.

> **An organization may not occupy a stage its control maturity does not support.** Level 2 across all dimensions is the entry condition for Stage 2. Level 3 is the entry condition for Stage 3.

Five levels across fifteen dimensions: the four mechanics domains in Part II, the seven lifecycle stages, and the four cross-cutting domains.

An organization's overall level is the **minimum across dimensions, not the average**. Agentic delivery fails at its weakest boundary, and an arithmetic mean conceals precisely the gap that matters—an organization with excellent development controls and no oracle architecture is not at Level 3, it is at Level 1 with a good development story.

18.1 Level Definitions

L0	**Ad hoc**	AI is in use; nobody knows where, by whom, or at what tier. No AI-specific controls. Policy, if it exists, is aspirational.
L1	**Aware**	Partial inventory. Policy drafted. Controls manual, inconsistent, applied only to centrally procured tooling. Measurement is anecdotal.
L2	**Governed**	Complete inventory of managed AI participation. Tiering applied. Identity, provenance, review, and evaluation baselines defined and enforced at deployment. Measurement exists and is human-paced.
L3	**Enforced**	Controls enforced by the pipeline rather than by policy. Ephemeral credentials standard. Evaluation gates merge and release. Provenance emitted and verified. Absorption capacity measured and governing.
L4	**Adaptive**	Continuous validation in production. Autonomy expansion driven by measured control efficacy. Independent assurance against evidence. Behavioral baselining per agent. Comprehension and maintainability managed as first-class operational concerns.

Realistic targets. Level 2 is the entry condition for Stage 2 and the minimum defensible bar for any AI participation at tier A2 or above in a repository that reaches production. Level 3 is the entry condition for Stage 3 and the target for organizations operating at scale or in a regulated sector. Level 4 is where autonomy expansion becomes genuinely data-driven.

18.2 The Grid

DIMENSION	L0	L1	L2 · MINIMUM BAR	L3	L4
M1 **Loop Governance**	Loops unbounded and uninstrumented	Limits set per team, inconsistently	Turn and budget ceilings on every A2+ deployment; termination typed	Early-abort detection; constraints verified to survive compaction; cost distributional	Loop telemetry drives autonomy decisions; divergence halted automatically
M2 **Work Class Governance**	Delegation is per-task habit	Classes named informally	Register with owners, tiers, and declared oracles	Oracles immutable and demonstrated; stop conditions halt; review cost funded	Class outcomes trended per language and repository; tiers adjust on evidence
M3 **Validation Architecture**	Tests only, generated alongside code	Independence stated as policy	Layered checks; oracle independent of generator	Oracle outside agent write scope; held-out signal; hardened evaluation boundaries	Mutation-gated; differential verification; visible-to-held-out gap monitored
M4 **Platform**	Licenses and guidance	Shared tooling, no substrate	Isolated runtime, governed tool gateway, agent identity registry	Entitlement-preserving context substrate; eval infrastructure; cost per task enforced	Fleet observability joined to outcomes; guardrails graduate from teams to platform

DIMENSION	L0	L1	L2 · MINIMUM BAR	L3	L4
S1 **Planning**	AI unpriced and unplanned	Tooling cost budgeted as licenses	Participation statement, tier, and consumption forecast per initiative	Absorption capacity modeled from telemetry and governing staffing	Autonomy and investment decisions driven by measured control efficacy
S2 **Requirements**	Prose requirements only	Acceptance criteria present, inconsistently	Machine-legible criteria; NFRs numeric; probabilistic thresholds defined	Criteria drive automated verification; traceability enforced end to end	Requirements quality measured against downstream defect and rework rates
S3 **Design**	Architecture tacit	Documented, not machine-surfaced	Constraints in a source-controlled artifact; blast radius documented; models pinned	Constraints enforced by blocking checks; environment separation verified in the running system	Architectural drift detected continuously; reuse and duplication actively managed
S4 **Development**	Agents on human credentials, unscoped	Distinct accounts, manual scoping	Unique agent identity; configuration reviewed as code; dependencies allowlisted; self-approval blocked	Ephemeral credentials; provenance signed at authorship; review risk-tiered and integrity-measured	Review integrity governs merge rate automatically; agent scope adapts to measured behavior
S5 **Verification**	Generated tests, generated code, same session	Tests reviewed by humans	Evaluation harness per model-backed capability; independence policy; qualification before write access	Evaluation gates merge and release; judges validated; toolchain red-teamed on a schedule	Statistical acceptance throughout; contamination-resistant corpora; qualification re-run on version change

DIMENSION	L0	L1	L2 · MINIMUM BAR	L3	L4
S6 **Release**	Authorship unknowable from the record	Manifest lists components	Release record distinguishes agent-authored change; SBOM produced; artifacts signed	Provenance verified as a gate; separation of duties tested; model changes gated as releases	Rollback on quality distribution automated with measured time-to-effect
S7 **Operations**	Aggregate billing, no agent telemetry	Logs collected, unanalyzed	Agent telemetry centralized; cost attributed; kill switches documented	Continuous validation against production; kill switches tested with measured time-to-effect; NHI review separate	Behavioral baselining; comprehension and maintainability funded from measured signal
X1 **Identity**	Shared or human credentials	Distinct accounts, manual lifecycle	One identity per deployment; documented lifecycle; registry with named owners	Task-scoped delegation; inspectable chains; automated deprovisioning	Continuous authorization; risk-adaptive scope
X2 **Provenance**	None	Attribution by convention, unverified	Signed commits with session linkage; SBOM at release	Attestation chain from source to release, verified at consumption	Attestation queryable across the estate; forensic reconstruction rehearsed
X3 **Evaluation**	None	Ad hoc, per team	Registry of corpora, metrics, thresholds, owners	Gating, validated judges, statistical acceptance, results as telemetry	Continuous validation; corpora derived from live traffic; efficacy measured
X4 **Governance**	None	Policy written	Review body with authority to block; named owners at A2+; exception register	Independent assurance; approval integrity measured; separate NHI reporting	Autonomy decisions data-driven; workforce and absorption managed as capacity

18.3 Using the Grid Honestly

Two instructions that determine whether this is a management tool or a slide.

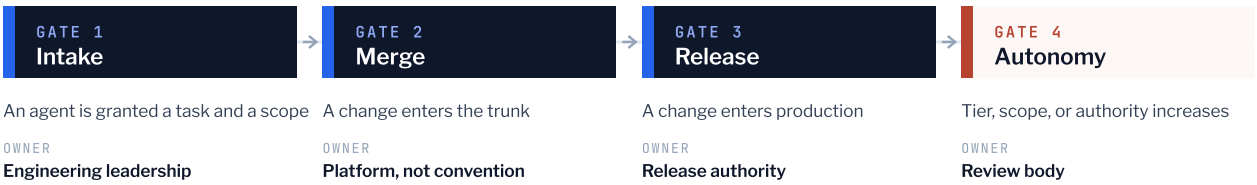
- **Assess against evidence, not intent.** Every cell above corresponds to an artifact named in the relevant section's evidence list. A dimension is at a level when the artifacts exist and can be produced, not when the practice is believed to be in place. The gap between those two states is the single most common finding in independent assurance.
- **Report the minimum and the distribution.** Report the overall level as the minimum, and separately show the profile across all fifteen dimensions. The profile is what tells leadership where the next investment goes; the minimum is what tells them what they can defend.

19 The Four Gates

Gates are the points where the framework's controls actually bind. Each is a pass-or-fail decision, not a scorecard—partial satisfaction is failure, because a gate that admits partial satisfaction is a checklist.

FIGURE 7

Where the controls bind



The four action boundaries defined in Section 2, expressed as gates with owners. Controls placed before a boundary are preventive; controls placed after it are forensic.

01 Gate One — Intake

Applied when an AI participant is granted a task and a scope. Owned by engineering leadership.

- [ ] Agent registered with a unique identity and a named accountable human
- [ ] Scope enumerated: repositories, paths, tools, environments, data classes
- [ ] No standing credentials in the execution context; credential lifetime documented
- [ ] Work class registered with a declared, demonstrated oracle that the agent cannot modify
- [ ] Entry and stop conditions defined for the work class, configured to halt rather than alert
- [ ] Concurrency limits set for the target repositories, with scope partitioned by path
- [ ] Autonomy tier assigned and approved at the required authority level
- [ ] Execution isolation configured; egress default-deny with an owned allowlist
- [ ] Qualification completed for the tier requested, with criteria and approver recorded
- [ ] Turn and budget ceilings configured; termination reasons emitted as typed telemetry
- [ ] Review cost for the class estimated and confirmed against available capacity
- [ ] Decommission criteria and review date defined

02 Gate Two — Merge

Applied to every change entering a protected branch. Enforced by the platform, not by convention.

- [ ] Change attributable to an identified principal; agent-authored change signed and session-linked
- [ ] Review performed at the tier the change's risk classification requires
- [ ] New dependencies allowlisted; versions resolved by policy and pinned with integrity hashes
- [ ] Static analysis and secret scanning blocking, with findings adjudicated rather than suppressed
- [ ] Held-out verification signal passed where the work class carries one; visible-to-held-out gap within threshold
- [ ] Architectural constraint checks passed, including duplication threshold
- [ ] Agent cannot approve its own change; enforcement verified, bypass actors reviewed
- [ ] Configuration, dependency manifest, and infrastructure changes reviewed as privileged code
- [ ] Concealed-character scan clean on configuration and instruction files
- [ ] Tests independent of the generating context; suite sensitivity verified by mutation or property testing
- [ ] Evaluation gate passed where the change touches a model-backed capability
- [ ] Requirement traceability identifier present

03 Gate Three — Release

Applied to every promotion into production, including model version changes with no code change. Owned by the release authority.

- [ ] Release manifest identifies agent-authored components and model versions
- [ ] Artifacts signed; provenance verified rather than merely emitted
- [ ] Non-functional gates passed: accessibility, performance, resource ceilings
- [ ] Rollback trigger defined on quality distribution, with measured time-to-effect
- [ ] Approval recorded and attributable to a named human
- [ ] SBOM produced, covering AI components where present
- [ ] Separation of duties preserved across generation, approval, and deployment; verified this quarter
- [ ] Evaluation thresholds met with intervals reported, not point estimates
- [ ] Operational readiness confirmed: telemetry, cost attribution, kill switch tested

04 Gate Four — Autonomy Expansion

Applied whenever an agent's tier, scope, or authority increases. Owned by the review body; never automatic.

- [ ] Operating evidence produced for the current tier: defect rate, rework rate, scope adherence, conflict rate, incident involvement
- [ ] Findings triage capacity sufficient at the projected volume
- [ ] Blast radius at the new scope enumerated and bounded
- [ ] Oracle strength adequate for the new scope, given that verification degrades with scope rather than with model quality
- [ ] Kill switch tested at the new scope with measured time-to-effect
- [ ] Termination conditions defined for the expanded scope
- [ ] Review integrity metrics within threshold for the affected repositories
- [ ] Absorption capacity modeled at the new volume and confirmed available
- [ ] Controls required by the new tier implemented and verified, not planned
- [ ] Reward-hacking indicators within threshold: visible-to-held-out gap, escalation rate, oracle-modification attempts
- [ ] Risk acceptance signed at the tier-appropriate authority, with a review date

The fourth gate is the one organizations most often skip, because expansion usually happens by accretion rather than by decision—a scope widened here, a repository added there, a bypass actor configured to unblock a deadline. Expansion by accretion is how an A1 deployment becomes an A3 deployment without anyone approving an A3 deployment.

20 Metrics and Executive Reporting

Report non-human and human identity governance separately, always. A combined figure is the single most misleading number in this domain, because good human hygiene conceals a much worse machine posture.

LOOP

- Termination reason distribution, trended—a population that is entirely "success" means the taxonomy is not instrumented
- Cost per task as a distribution, with the ninety-fifth percentile, not a mean
- Trajectory length for successful versus failed sessions
- Early aborts fired, and cost recovered
- Escalation rate per agent-week—suppression of escalation is a defect, not efficiency

WORK CLASS

- Merge, revert, and defect-escape rate by class, against a per-language and per-repository baseline
- Declared review cost against measured review hours, by class
- Classes throttled or tier-reduced on evidence in the trailing year
- Visible-suite to held-out-suite pass rate gap, by class—the reward-hacking indicator

COVERAGE

- Registered AI participants versus discovered AI participants; the gap is the metric
- Share of the agent estate emitting action telemetry
- Share of A2-and-above deployments with a named accountable human
- Share of repositories with enforced merge-gate configuration

FLOW AND ABSORPTION

- Change volume, split by agent-authored and human-authored
- Median and 95th-percentile time in review, trended
- Share of changes merged without human review
- Review hours required versus senior engineering hours available—the absorption ratio
- Merge conflict rate, with cross-agent concurrency broken out

QUALITY

- Change failure rate and incidents per unit of change, trended against AI adoption
- Defect escape rate by authorship class
- Security findings generated versus adjudicated; suppression and threshold changes reported explicitly
- Duplication and reuse trend
- Non-functional conformance: accessibility, performance budget adherence

VERIFICATION INTEGRITY

- Share of model-backed capabilities with a current, refreshed evaluation corpus
- Judge validation currency and chance-corrected agreement
- Mutation score for agent-authored modules, reported alongside line coverage
- Toolchain red team attack success rate, trended

CONTROL EFFICACY

- Median credential lifetime in agent execution contexts
- Share of agents with zero standing credentials
- Share of releases with verified provenance
- Separation-of-duties test results, by quarter
- Kill-switch time-to-effect by agent class, from live test

PLATFORM

- Context substrate coverage: share of repositories indexed and cataloged, entitlement enforcement verified
- Share of tool access flowing through the governed gateway rather than point integrations
- Agent telemetry generated against telemetry analyzed
- Guardrails graduated from team-local to platform capability

WORKFORCE

- Entry-level hiring and internal promotion rate, trended over eight quarters
- Senior engineer tenure and attrition, read alongside review load
- Seeded-defect probe catch rate and median review time
- Comprehension: subsystems with a named owner who can explain them

ECONOMICS

- Cost per task and cost per merged change, trended
- AI consumption against forecast, with variance explained
- Total cost per engineer including consumption, against the license line

GOVERNANCE

- Deployments by autonomy tier, with trend
- Deployments blocked or tier-reduced at review
- Active exceptional-access grants and their expiry dates
- Approval rate and median review duration for human-in-the-loop controls

TWO LEADING INDICATORS WORTH WATCHING ABOVE THE OTHERS

First, **agent population growth rate against registry coverage growth rate**: when the first exceeds the second, governance is degrading regardless of what any other metric says. Second, **the absorption ratio**: when required review hours exceed available senior engineering hours, every downstream quality metric will follow within two quarters, and the correct response is to reduce the merge rate rather than to exhort the team.

21 Tooling Evaluation Criteria

This framework is deliberately vendor-agnostic, and this section is where that posture becomes actionable.

Two distinct question sets are needed and they are usually collapsed into one. The first is universal: a set of governance properties that any AI participant in delivery must satisfy regardless of what stage it serves, because they determine whether the organization can attribute, bound, revoke, and audit what the tool does. The second is stage-specific: the properties that actually differentiate two tools serving the same lifecycle stage, and which look entirely different for a requirements platform than for a pipeline agent. A procurement process that asks only the universal questions buys governable tools that do not fit the work. One that asks only the stage-specific questions buys capable tools that cannot be governed. Ask both.

21.1 Capability Classes by Stage

Distinguish these classes, because they carry different risk, bind at different stages, and are frequently bundled and sold as a single product.

CLASS	STAGES	WHAT IT DOES	PRIMARY RISK
Planning and portfolio intelligence	S1	Forecasting, capacity modeling, work decomposition, consumption attribution	False precision in estimates; consumption reported after the fact rather than forecast
Requirements and specification	S2	Authoring, structuring, traceability, acceptance criteria generation	Generated specifications that are untraceable; criteria that no machine can check
Design and architecture	S3	Modeling, decision records, threat modeling, constraint definition, component catalogs	Architecture agents cannot read; constraints that exist only as documentation
Context and knowledge substrate	S2–S7	Indexes internal code, documents, and decisions, and exposes them to agents	Retrieval scoped wider than the invoking principal; poisoned or stale context
Inline assistant	S4	Completion and chat within the editor; no write authority beyond the developer's own actions	Quality and comprehension; licensing exposure
Repository-resident agent	S4	Operates on a repository, opens branches and pull requests, runs tools	Authorization, provenance, review saturation

CLASS	STAGES	WHAT IT DOES	PRIMARY RISK
Pipeline agent	S4–S6	Operates within CI/CD; builds, tests, triages, remediates	Credential blast radius; verification independence
Review and analysis agent	S4–S5	Reviews changes, triages findings, suggests remediation	False assurance; triage suppression; ingesting untrusted content
Test and evaluation	S5	Evaluation harnesses, evaluation as a gate, red teaming, test generation	Metric non-portability; self-consistent suites; unvalidated judges
Release and provenance	S6	Signing, attestation, SBOM generation, release orchestration	Provenance emitted but never verified; AI components absent from the SBOM
Operations agent	S7	Acts on running systems: deploys, remediates, scales, responds	Production blast radius; unreliable self-report
Observability and cost	S7	Telemetry, evaluation in production, consumption attribution	Recording mistaken for detection; cost visible only in aggregate
Agent platform / control plane	All	Governs which agents run, with what models, under what policy	Concentration of authority; enforcement gaps between platforms

Two observations from this table are worth carrying into a buying decision. Most enterprise attention and most vendor investment sit in the generation-heavy classes (inline assistants, repository-resident agents, and pipeline agents), while the evidence in Section 3 places the binding organizational constraint in the two classes that absorb their output, review and analysis and test and evaluation. And the first four classes, which govern what everything downstream is actually working from, are the least likely to be evaluated as AI tooling at all, because they were usually bought years ago for other reasons.

21.2 Universal Criteria

These apply to every class in the table above and are scored pass or fail, not weighted. A tool that fails several is ungovernable at any stage, however well it performs. Three carry evidence from elsewhere in this framework: automatic attribution that a user cannot correct corrupts the record it creates, which is one reason at least one major editor ships its equivalent setting defaulting to off;[16] licensing exposure from training-data reproduction is small but non-zero and scales with context size, which is the direction agentic tooling is moving;[49,50] and the measured cross-product merge conflict rate is roughly double the same-product rate, so multi-vendor strategies carry a cost no vendor comparison will surface.[32]

ID	CRITERION	A STRONG ANSWER CONTAINS	A WEAK OR EVASIVE ANSWER
U-1	**Identity and offboarding**	Unique principal per deployment; delegation chain inspectable after the fact; identities governable in your IdP as enforcement, not inventory; a described offboarding process	“Agents run as a service account”; governance described as visibility; no answer at all on offboarding
U-2	**Credential model**	Task-scoped, short-lived credentials; measured revocation time-to-effect	“Credentials are stored securely”; revocation described but never measured
U-3	**Isolation and egress**	Isolation verified in the running environment; every outbound path enumerated; artifact proxy inside production vulnerability management; untrusted-input paths separated from privileged ones	Isolation asserted from an architecture diagram; egress described as “restricted” without enumeration
U-4	**Configuration surface**	Named list of files treated as executable or authoritative; approval bound to content hash; concealed characters filtered; defined behavior on a hostile repository	Configuration described as inert; approval bound to a server or tool name
U-5	**Provenance and attribution**	Cryptographically verifiable attribution; model version recorded; exportable session record linking artifact to task, human, and tool set; attribution configurable with a stated default	Attribution present but unsigned; no model version; session records visible in a console but not exportable
U-6	**Model dependency**	Dated snapshot pinning; stated notice commitment before version or safety-configuration change	Alias-only model selection; “we notify customers of material changes”
U-7	**Data handling and IP**	Training use stated with the default named; retention period and jurisdiction stated; indemnity with exclusions disclosed; license matching surfaced at suggestion time	Training use described as “may be used to improve the service”; indemnity referenced without its exclusions

ID	CRITERION	A STRONG ANSWER CONTAINS	A WEAK OR EVASIVE ANSWER
U-8	**Telemetry and control**	Emission in a documented schema ingestible by your pipeline; measured kill-switch time-to-effect; central administration of approved models and permitted agents	Telemetry available only in the vendor console; kill switch that reaches orchestrated agents but not local ones
U-9	**Concurrency and interoperability**	Described behavior under concurrent agents; scope partitioning supported; conflict behavior documented	"Our agents coordinate automatically"; no position on other vendors' agents
U-10	**Exit**	Named export path for evaluation corpora and metric definitions, provenance records, session logs, agent definitions, instruction files, policy configuration, traceability identifiers	Data export offered for content but not for governance artifacts
U-11	**Assurance and incident commitment**	Mapping to a published agentic risk taxonomy with a stated shared-responsibility line; independent assessment of the agentic components specifically; a notification commitment and named incident support terms	A platform-wide certification offered in answer to an agent-specific question

21.3 Stage-Specific Criteria

These are weighted rather than pass-fail, because their relative importance depends on where the organization's constraint sits. Score each 2 (met and demonstrated), 1 (partially met or met by roadmap), or 0 (not met). An unanswered criterion scores 0; the absence of an answer is data.

2	1	0
Met and demonstrated	Partially met, or met by roadmap	Not met — or unanswered

Three criteria below rest on evidence stated earlier: between 36.7 and 55.7 percent of tasks in one study carried a known CVE in the dependency version the model specified, which is why S4-2 exists;[64] the capabilities that most determine whether an organization realizes value are organizational rather than product properties, which is why S1-4 is a caution rather than a criterion;[98] and published guidance exists on the security content that belongs in agent instruction files, which is what S3-5 scores against.[99]

S1 Planning

ID	CRITERION	A STRONG ANSWER CONTAINS	A WEAK OR EVASIVE ANSWER
S1-1	**Consumption attribution**	Cost attributable to team, repository, and task	Attribution to a billing account only
S1-2	**Consumption forecasting**	Forward forecast with variance against actuals	Reporting after the period closes
S1-3	**Absorption modeling**	Review load modeled alongside delivery throughput	Throughput dashboards with no review-side counterpart
S1-4	**Honest scope**	Vendor distinguishes what the product supplies from what the organization must supply	Product claimed to deliver organizational capabilities such as an AI stance or platform quality

S2 Requirements

ID	CRITERION	A STRONG ANSWER CONTAINS	A WEAK OR EVASIVE ANSWER
S2-1	**Machine-legible criteria export**	Structured export a test generator or evaluation harness consumes without the vendor in the loop	Requirements exported as prose or PDF
S2-2	**Traceability durability**	Identifier survives into branches, commits, tests, and release records in systems the vendor does not own	Traceability maintained only inside the vendor's own graph
S2-3	**Generated-requirement marking**	Drafts marked as generated with the model recorded	Generated and authored requirements indistinguishable after save
S2-4	**Non-functional expression**	NFRs expressible as numeric budgets with named owners	NFRs captured as free-text intentions
S2-5	**Probabilistic requirements**	Threshold, corpus reference, and measurement method as first-class fields	Model-backed behavior described in the same fields as deterministic behavior

S3 Design

ID	CRITERION	A STRONG ANSWER CONTAINS	A WEAK OR EVASIVE ANSWER
S3-1	**Enforceable constraint output**	Constraints emitted in a form the build can enforce	Constraints rendered as diagrams and documents only
S3-2	**Agent-legible design surface**	Design representation surfaceable to a coding agent as authoritative context	Design accessible only through the vendor's UI
S3-3	**Artifact governance**	Constraint and instruction artifacts live in version control, reviewable	Artifacts held as platform configuration outside review
S3-4	**Reuse discovery**	Component catalog answers "does something like this already exist"	Search over names rather than over capability
S3-5	**Instruction-file security content**	Instruction files carry dependency, package-manager, pinning, and integrity-verification guidance	Instruction files treated purely as style configuration
S3-6	**Threat model comparability**	Findings mapped to a published taxonomy	Bespoke severity language, not comparable across teams

S4 Development

ID	CRITERION	A STRONG ANSWER CONTAINS	A WEAK OR EVASIVE ANSWER
S4-1	**Merge boundary, demonstrated**	Live demonstration that the agent cannot self-approve, mark its own PR ready, trigger CI unauthorized, push to protected branches, or edit its governing ruleset	Documentation offered in place of a demonstration
S4-2	**Dependency version resolution**	Versions resolved against vulnerability policy, not accepted as the model specified them	Model-specified versions written directly into the manifest
S4-3	**Package existence and provenance checks**	Existence, age, and maintainer provenance checked before install	Checks run after install, or at scan time only
S4-4	**Risk-tiered review**	Review requirements assignable by path and change class	Uniform review policy with per-repository granularity only
S4-5	**Review integrity telemetry**	Approval rate, review duration, and diff size at approval reported	Merge counts reported without review-quality signal
S4-6	**Concealed-character handling**	Ingested content filtered for zero-width and bidirectional characters	No stated position on invisible-rendering content

S5 Verification

ID	CRITERION	A STRONG ANSWER CONTAINS	A WEAK OR EVASIVE ANSWER
S5-1	**Evaluation as a local gate**	Evaluations run inside your CI without a vendor callback	Gating available only through the vendor's hosted pipeline
S5-2	**Metric transparency and portability**	Documented metric computation; corpora and results exportable in an open format	Named metrics with undisclosed computation
S5-3	**Judge validation**	Validation against human labels supported as a first-class workflow, with chance-corrected agreement reported	Judge scores presented as ground truth
S5-4	**Suite sensitivity**	Mutation or property-based testing establishes that a suite would fail on a defect	Line coverage reported as the quality signal
S5-5	**Remediation verification independence**	The verifier of a suggested fix is independent of the model that produced it, with false-negative rates disclosed	Time-to-fix improvements published; false negatives not measured

S6 Release

ID	CRITERION	A STRONG ANSWER CONTAINS	A WEAK OR EVASIVE ANSWER
S6-1	**Signed provenance**	Attestations in a standard predicate format, verified at consumption	Provenance emitted but never checked
S6-2	**Agent-authorship predicate**	Support for a custom predicate carrying agent authorship metadata	Authorship recorded only in commit trailers
S6-3	**AI-inclusive SBOM**	SBOM covers AI components including model lineage and dataset provenance	SBOM covers packages only
S6-4	**Authorship in the release record**	Agent-authored and human-authored change distinguishable without manual reconstruction	Distinction reconstructable only by querying commit history by hand
S6-5	**Quality-distribution promotion gates**	Promotion gated on evaluation score, refusal rate, or task completion distribution	Promotion gated on error rate alone
S6-6	**Model version change as a release**	Provider version change traverses the release gate	Model version treated as runtime configuration

S7 Operations

ID	CRITERION	A STRONG ANSWER CONTAINS	A WEAK OR EVASIVE ANSWER
S7-1	**Schema-versioned telemetry**	Published schema you can pin and version, given these conventions remain development-stage	Proprietary event shapes that change without notice
S7-2	**Unit cost observability**	Cost attributable per task and per agent	Account-level billing only
S7-3	**Kill-switch reach**	Halt reaches agents in developer environments and third-party platforms, with measured time-to-effect	Halt reaches orchestrated agents only
S7-4	**Intra-snapshot drift detection**	Behavioral change detected within a pinned snapshot	A deprecation calendar offered as drift detection
S7-5	**Continuous validation**	Evaluation suite runs against production traffic on a schedule with distribution alerting	Evaluation available pre-release only

21.4 Scoring and Use

- **Run U-1 through U-11 first, as gates.** They are pass or fail. A vendor failing three or more should not proceed to stage scoring, because the stage capability will not survive the governance gap.
- **Score the stage criteria only for the stages the tool actually serves,** using the capability-class table in 16.1 to decide which apply. Scoring a requirements platform against S6 produces a low number that means nothing.
- **Weight stage fit by where your constraint actually is.** Most organizations shop for stage four generation speed. The evidence in Section 3 places the binding constraint in stage five verification capacity and in the review load that stage four generates. Weights that mirror the evidence rather than the sales motion produce a different shortlist, which is the point of scoring at all.
- **Weight the universal criteria above capability.** Capability improves rapidly and converges across products; governance architecture is set at design time and rarely changes. A tool that generates excellent output and cannot be governed will be replaced within eighteen months at considerable cost. A tool that generates adequate output inside a sound authorization model will still be serviceable.
- **Resolve the suite-versus-best-of-breed question deliberately.** The concurrency finding argues for fewer agent products operating on the same repository. The portability and exit criteria argue against a single vendor owning the measurement layer, because a vendor that both generates and grades is not an independent verifier. The reconcilable position is to consolidate the things that write and diversify the things that measure.

- **Demand demonstrations for U-1 through U-5 and for S4-1.** All six are testable in an afternoon, and all six are areas where vendor documentation and shipped behavior have diverged in public. Score a documented-but-undemonstrated claim as 1, never 2.
- **Record vague answers as findings rather than as zeros.** A zero says the capability is absent; a finding says the vendor could not describe its own product's behavior, which predicts the support relationship. The pattern worth watching for is a confident answer about the platform's general security posture offered in response to a specific question about agent behavior.

22 Standards Crosswalk

Practice identifiers in the SSDF column follow Secure Software Development Framework version 1.1, which remains the operative final version; a draft revision was released for comment in December 2025 and had not been finalized as of August 2026.[97] Note that the generative AI community profile reintroduces practice PW.3 with an entirely new meaning—confirming the integrity of training, testing, fine-tuning, and aligning data—where SSDF 1.1 has no PW.3 at all.[10] Citing PW.3 without naming the document is a common and avoidable error.

THIS FRAMEWORK	NIST SSDF (800-218 / 218A)	ISO/IEC 42001 / 5338	EU AI ACT	OTHER
S1 **Planning**	PO.1, PO.2	42001 Cl. 6, A.2, A.3; 5338 organizational project-enabling processes	Art. 9 (risk management)	CRA Art. 13; FinOps Framework
S2 **Requirements**	PO.1, PW.1	42001 A.6 (requirements and specification); 5338 stakeholder and system requirements definition	Arts. 9, 10, 13	ISO/IEC 27002 8.26
S3 **Design**	PW.1, PW.2, PW.4	42001 A.6 (design and development); 5338 architecture and design definition	Arts. 9, 14, 15	ISO/IEC 27002 8.27; MITRE ATLAS; OWASP Agentic Top 10
S4 **Development**	PW.5, PW.6, PW.7, PS.1	42001 A.6; 5338 implementation	Art. 15	ISO/IEC 27002 8.28, 8.31; PCI DSS 6.2; SLSA Source Track; OWASP NHI Top 10
S5 **Verification**	PW.7, PW.8, RV.1	42001 A.6 (verification and validation); 5338 verification, validation, continuous validation	Arts. 9, 15	ISO/IEC 27002 8.29, 8.33; NIST AI 800-2; NIST AI 100-2e2025
S6 **Release**	PS.2, PS.3, PW.9	42001 A.6 (deployment); 5338 transition	Arts. 11, 12, 16	ISO/IEC 27002 8.32; SLSA Build Track; CISA SBOM minimum elements; CRA Annex I
S7 **Operations**	PO.5, RV.1, RV.2, RV.3	42001 A.6 (operation and monitoring, event logging); **5338 continuous validation**	Arts. 12, 15, 26	CRA Art. 14 reporting; SOC 2 CC7; OTel GenAI conventions
X1 **Identity**	PO.2, PO.5	42001 A.3, A.4	Art. 14	OWASP NHI Top 10; NCCoE agent identity project

THIS FRAMEWORK	NIST SSDF (800-218 / 218A)	ISO/IEC 42001 / 5338	EU AI ACT	OTHER
X2 **Provenance**	PS.1, PS.2, PS.3, PW.4	42001 A.10 (third-party relationships)	Arts. 11, 12	SLSA; in-toto; Sigstore; CycloneDX / SPDX; CISA AI SBOM elements
X3 **Evaluation**	PW.8, RV.1	42001 A.6; 5338 continuous validation	Arts. 9, 15	NIST AI 800-2; NIST AI RMF MEASURE
X4 **Governance**	PO.1–PO.4	42001 Cl. 4–10, A.2, A.3, A.5	Arts. 14, 17, 26	SOC 2 CC1–CC5, CC8; DORA RTS Arts. 15–17; ISO/IEC 42005

22.1 Regulatory Notes

Four points where a framework drafted earlier in 2026 would now be wrong, and one where the underlying assumption is worth stating.

The EU AI Act's high-risk obligations have been deferred. Regulation (EU) 2026/1744, the Digital Omnibus on AI, was adopted on July 8, 2026 and entered into force on July 27, 2026.[94] It moved the Chapter III high-risk obligations for Annex III stand-alone systems from August 2, 2026 to **December 2, 2027**, and for Annex I product-embedded systems from August 2, 2027 to **August 2, 2028**, citing delayed availability of harmonised standards and delayed designation of national competent authorities. As of August 2026 the binding obligations are the prohibited practices and AI literacy requirements in force since February 2025, the general-purpose AI obligations in force since August 2025, and the Article 50 transparency obligations in force since August 2, 2026. Any framework asserting that Articles 9 through 15 currently bind Annex III providers is wrong by roughly sixteen months. Plan against the deferred dates; do not claim they are current.

The US federal software attestation mandate was rescinded. OMB Memorandum M-26-05, issued January 23, 2026, rescinded the memoranda that created the government-wide secure software self-attestation requirement.[95] The attestation form and the SSDF remain available and are now optional, with agencies setting tailored risk-based requirements. The SSDF survives as guidance rather than as a procurement condition. This is a material change for anyone whose control rationale rested on a federal mandate.

The Cyber Resilience Act's reporting obligations begin imminently. Reporting for actively exploited vulnerabilities and severe incidents applies from September 11, 2026, with full application on December 11, 2027.[46] Manufacturers must maintain an SBOM covering at least top-level dependencies, publish a coordinated vulnerability disclosure policy, and define a support period. It is the first regime to make secure development practice a market access condition.

Sector regimes were drafted without agents in contemplation, and the regulator's position will be that existing obligations apply unchanged. The DORA regulatory technical standards require testing and approval of all systems before production use, source code review covering both static and dynamic testing, non-production environments containing only anonymised or pseudonymised production data, and independence between the approving and

implementing functions.[4] PCI DSS requires pre-release review of bespoke and custom code, and where that review is manual, a reviewer other than the originating author; automated review is permitted as an alternative.[2] The FDA's predetermined change control plan is the only instrument among these that pre-authorizes a bounded envelope of future autonomous change, and it is worth studying as the closest available regulatory precedent for governing systems that modify themselves after release.[96]

The underlying assumption. Where an agent acts on regulated data or systems, document the mapping explicitly rather than waiting for guidance. Regulators are unlikely to accept that an obligation lapsed because the party performing the work was not a person.

23 Open Problems

A framework is more credible for naming what it cannot yet answer. These are genuine gaps in standards, practice, or evidence—not areas awaiting a vendor. Several will close on events that are already scheduled or pending; Appendix C tracks those separately, so that a reader can tell how stale this document has become without reading it twice.

- **No standard exists for attesting AI authorship.** Neither major SBOM format has a native construct for "this artifact was generated by an agent." What exists is an unsigned commit-trailer convention, platform-specific attribution metadata that has demonstrated it can be wrong, and source provenance that records how a revision was created without a field for what generated its content. The custom predicate proposed in X2 is a workaround, not a solution.
- **No accepted criterion governs granting an agent write access.** Benchmarks measure capability on curated tasks; they measure nothing about behavior under organizational constraint, and the gap between near-saturated benchmark scores and observed merge and security rates makes them actively misleading as a qualification signal. The staged qualification protocol in Section 14 is defensible and it is invented.
- **No standard governs rollback on behavioral change.** Progressive delivery mechanics are mature; the trigger for a model-backed feature that degrades without erroring is not defined anywhere, and there is no independent research on it.
- **No error budget model works when correctness is unobservable at request time.** The observable-proxy approach in Section 16 is a workaround. The conceptual problem, that a budget cannot be debited for failures that are not detected, is unsolved.
- **No published contractual practice covers behavioral stability within a pinned snapshot.** Providers publish deprecation notice periods. None commits to behavioral stability for a snapshot that remains nominally available, which is the risk that actually breaks running systems.
- **No independent data exists on static analysis efficacy against AI-generated code specifically.** Every efficacy figure available is vendor-published, and none discloses false-negative rates, which is the number that matters for a triage automation.
- **No corpus of AI incident postmortems exists.** Two or three public incidents are cited repeatedly across the entire industry because they are nearly the only ones published. Conventional outage postmortems became a shared engineering resource through a culture of publication that has not yet formed here.
- **No research addresses performance or load testing under AI-assisted development.** The accessibility evidence is real and independent; there is no equivalent for performance, and inferring one from reuse decline would be reasoning past the data.
- **Comprehension debt is a concept, not a measurement.** It names something real and widely observed. There is no instrument for it, no baseline, and no longitudinal study—only a small randomized trial on skill formation and population-level maintainability trends that are correlational.

- **No published data establishes an engineer-to-agent supervision ratio.** Not from a vendor, not from an enterprise, not from academia. The term originates in a vendor survey that explicitly declined to supply a figure. Absorption capacity is measurable and should be measured in its place.
- **Sub-agent context isolation has a documented mechanism and an asserted benefit.** No published measurement exists of whether it improves task success on software engineering work. The same is true of checkpoint-and-restart efficacy, git-worktree parallelism, and plan-and-act mode separation.
- **Spec-driven agentic development has tooling and no efficacy data.** No benchmark, no user study, no independent evaluation. Test-driven agentic development, by contrast, is peer-reviewed and works.
- **Build and CI repair has an excellent oracle and no published enterprise deployment data.** It is the highest-prospect unevidenced work class on the list.
- **Documentation maintenance by agents has no published enterprise outcome data at all**, and no oracle. It is simultaneously the most-delegated and the least-measured work class.
- **No published deployment gates agent-written tests on mutation score.** It follows directly from two documented enterprise mutation-testing programs and nobody has published doing it.
- **Shadow-mode deployment of agent output is unmeasured.** The pattern is well documented pre-AI; no enterprise has published applying it to agent changes.
- **No enterprise has published its agent-fleet observability architecture, its sandbox compute spend, or measured savings from production model routing.** The one large published fleet-level token saving came from response shaping at a tool gateway, not from routing.
- **No published study measures human review effectiveness on agent-authored code against a ground-truth defect set**, nor rubber-stamping rates in agent pull request review. Both are load-bearing for any auto-merge policy.
- **No published study measures skill formation in juniors trained primarily under agentic workflows.** Strong indirect evidence, zero direct longitudinal evidence.
- **No validated maturity model for agentic software engineering exists.** A rigorous diagnostic, a credible general AI maturity model, and a proposed governance escalation exist; a validated staged model of agentic engineering adoption does not. This framework's Section 4 is a synthesis, and is offered as one.
- **No model exists for when a temporal control substitutes for a logical one.** One organization automerges the large majority of millions of maintenance changes with no per-change human verification, relying on cohort gating and post-deployment failure correlation rather than on deciding correctness before the merge (Section 38). It publishes no failure or rollback rate, so the safety of the design is asserted rather than demonstrated —and yet it has operated for years. The conditions under which detect-and-stop is an adequate substitute for verify-then-merge, and the blast-radius threshold at which it stops being one, are unmodeled here and unstudied anywhere.
- **Standards bodies have not yet arrived.** NIST has an initiative, a request for information, a concept paper, an annotated outline, and a preliminary draft—no normative text on agents. The two planned control overlays for single-agent and multi-agent systems do not exist in draft form. This framework is written for that interval and should be revised when they publish.

PART V

Practitioner Playbooks

Ten procedures, each producing a specific artifact. Where a practice rests on measured evidence, the closing note of the chapter says which. Where it rests on reasoning because no published measurement exists, the chapter says that instead. Both kinds appear here, and the distinction is never blurred.

24 Designing an Oracle

Produces. A written oracle specification for one work class: what "correct" means, what mechanism decides it, why that mechanism is independent of the agent, and what the agent cannot see.

Run this when. Before any work class moves above tier A1, and again whenever its scope widens. Section 6.1 puts oracle strength as the strongest predictor of delegation success. This is the chapter that turns that finding into an artifact.

24.1 The Three Properties

An oracle that lacks any one of these is not an oracle. It is a metric the agent will optimize.

Independent. Derived from the requirement rather than from the implementation, and not produced by the same model in the same session. When one model writes both the code and the test, a passing suite establishes self-consistency and nothing else.

Immutable. The agent cannot modify the oracle. This is a filesystem permission and a branch protection rule, not an instruction. Every verification signal an agent can reach becomes a target, and the measured remedy is architectural: removing write access to test and grading artifacts, combined with hardened evaluation boundaries, cut exploit rates by 87.7 percent relative with task success essentially unchanged.[113]

Partly hidden. A held-out signal the agent cannot observe during the run. This is the only reliable defense against optimization toward the visible check, and its absence is measurable: the gap between visible-suite and held-out-suite pass rates grows roughly 27 percentage points per tenfold increase in code size.[114]

24.2 Oracle Classes, Ranked by Strength

CLASS	MECHANISM	STRENGTH	COST	AGENT-VISIBLE
Compilation and type checking	The build succeeds and types resolve	Weak alone, free, catches structural error	Negligible	Yes

CLASS	MECHANISM	STRENGTH	COST	AGENT-VISIBLE
Execution against authored tests	A test suite written independently of the change passes	Moderate; strong when the suite is human-authored and immutable	Low	Yes, and therefore gameable
Held-out compositional tests	A suite the agent never sees, exercising features in combination	**Strong**	Medium	**No**
Differential comparison	Behavior matches a reference implementation or prior version	**Strong** where a reference exists	Medium	No
Mutation score	Injected defects are detected by the suite	**Strong** for validating agent-written tests	High, tractable when diff-scoped	Should not be
Property and invariant checks	Stated invariants hold across generated inputs	Moderate to strong	Medium	Partial
Statistical threshold on a corpus	Score distribution clears a threshold with a reported interval	Moderate; the only option for probabilistic components	Medium	Partial
Model-as-judge	A second model scores the output	**Weak unless validated against human labels**	Low	No
Human review	A person decides	Strong on intent and design, weak on mechanical defect, does not scale	Highest	No

Two entries deserve a warning. **Coverage is not on this list**, because a replicability study across 101,123 test cases and eleven models found coverage unreliable for real-fault detection precisely when the code under test contains bugs, which is the agent case.[179] And **model-as-judge belongs at the bottom** until validated: judges show a 33 to 41 percentage point gap between raw agreement and chance-corrected agreement, and two production judges exhibited test-retest reliability above 0.95 while carrying position bias above 0.10, meaning they were consistently wrong in the same direction.[78]

24.3 The Procedure

Step 1. State the correctness claim in one sentence. Not "the code works." Something falsifiable: *this change preserves the public behavior of module X while eliminating the deprecated call*. If you cannot write the sentence, you cannot build the oracle, and the work class is not ready for delegation.

Step 2. Choose the strongest affordable class. Work down the table until you reach something you can actually run on every change. Most work classes land on execution against authored tests, and most should add a second layer.

Step 3. Establish independence. Answer three questions in writing. Who wrote the oracle? Was it derived from the requirement or from the implementation? Can the same agent session that produces the change also produce or alter the check? If the answer to the last is yes, stop and fix that before proceeding.

Step 4. Make it immutable. Enumerate the paths the oracle lives in: test directories, fixture data, grading scripts, CI configuration, policy files. Remove agent write access to all of them. Then verify by attempting a write from a test agent and confirming it fails. A policy statement is not this step; the attempted write is.

Step 5. Build the held-out signal. Reserve a portion of the verification that never enters the agent's context or repository. Three workable constructions: a compositional suite exercising features together while the visible suite exercises them in isolation; a rolling holdout of recent production cases the agent has not seen; or a differential check against a reference the agent has no access to. Run it after the agent completes, never during.

Step 6. Calibrate before gating. Run the oracle against known-good and known-bad changes. Record its false-positive and false-negative rate. An oracle gating merges without a measured false-negative rate is a gate whose reliability you are asserting.

Step 7. Instrument the gap. Record visible-suite and held-out-suite results separately for every change, forever. The gap between them is your reward-hacking indicator, and almost no organization computes it.

24.4 Worked: Three Work Classes

Test generation and coverage backfill. Correctness claim: *this test fails when the behavior it describes is broken*. Primary oracle is mutation score rather than coverage, on the evidence that mutation-guided generation killed mutants at roughly six times the rate of its coverage-guided predecessor, 15 percent against 2.4 percent, and that 49 percent of the useful tests it produced added no line coverage at all.[170] Independence: the mutants are generated by the platform, not by the authoring agent. Immutability: the mutation configuration and the mutant corpus sit outside agent scope. Held-out: a mutant subset withheld from the reported score. Gate: a mutation-score floor, not a coverage floor.

Dependency upgrade with breakage repair. Correctness claim: *the application's observable behavior is unchanged and the vulnerable version is gone*. Primary oracle is execution against the existing authored suite, plus a differential check on public API surface. Independence is inherent, because the suite predates the change. Immutability is the critical step here and the most commonly skipped: the agent must not be able to edit tests to accommodate the

upgrade. Held-out: an integration suite the agent's branch does not run. Guard for the known failure mode, which is a resolved version the model chose rather than policy chose, given that 36.7 to 55.7 percent of tasks in one study carried a known CVE in the model-specified version.[64]

Documentation maintenance. Correctness claim: *the described behavior matches the code*. **This class has no oracle in most organizations, and that is the finding rather than an inconvenience.** Building one is possible and is the precondition for delegating above A1: executable examples that run in CI, reference and link validation, signature extraction compared against the documented signature, and a staleness check tying each document to the commit range of the code it describes. Until at least two of those exist, keep the class at A1 with mandatory human review. The reason is the risk shape, not squeamishness: generated documentation is well formed, authoritative-looking, and unverifiable, which is the profile in which readers cannot detect error.[175]

24.5 Instrumentation

- Visible-suite to held-out-suite pass rate gap, per work class, trended
- Oracle modification attempts, logged and alerted, target zero
- False-negative rate from calibration, refreshed quarterly
- Escalation rate per agent-week, where suppression of escalation is a defect rather than efficiency
- Share of merged changes gated by an oracle stronger than compilation

24.6 Failure Signatures

- **The gap widens quietly.** Visible pass rates hold steady while held-out rates decline. This is the signature of optimization toward the visible check and it is invisible without step 7.
- **Escalation goes to zero.** An agent that never says a task is impossible has either become perfect or has learned that producing something scores better than declining. Giving an agent an explicit impossibility affordance cut cheating from 54 to 9 percent for one model, so the affordance should exist and its use should be expected.[112]
- **The oracle starts failing on legitimate change.** A brittle oracle trains the organization to bypass it. Rising exception requests are a signal to fix the oracle, not to grant the exceptions.
- **Acceptance measured with the answers available.** If verification runs with network access and full git history, acceptance may be measuring retrieval rather than engineering: an audit of 731 successful trajectories found 63 percent of resolutions retrieved rather than derived, with scores dropping 14 to 21 points once history and network were restricted.[125]

24.7 What This Rests On

The three properties are measured, not asserted: independence from the reward-hacking mitigation result,[113] the held-out requirement from the scope-scaling finding,[114] and the ranking of judge and coverage oracles from the judge-reliability and coverage-correlation studies.[78,179] The seven-step procedure is this framework's construction, and the specific step sequence has not been validated in the field.

25 Loop Instrumentation

Produces. An event schema for agent sessions, a small set of derived metrics, and thresholds calibrated from your own data rather than borrowed from a publication.

Run this when. Before raising any deployment above A1, and in the first thirty days of any program regardless of stage. It is the cheapest diagnostic available and most organizations have none of it.

25.1 The Minimum Event Set

Six events. Anything less cannot answer why a run cost what it cost or why it stopped.

EVENT	REQUIRED FIELDS	ANSWERS
`session.start`	agent identity, autonomy tier, work class, model and version, invoking human, task reference, scope	Who authorized this, against what, under which model
`turn.complete`	turn index, input tokens, output tokens, elapsed, tool calls issued	Where cost accumulates within a run
`tool.call`	tool name, target, parameters hash, outcome, duration	What the agent actually did, and what was denied
`context.compact`	turn index, tokens before and after	When instructions may have been summarized away
`session.end`	**typed termination reason**, total cost, turns, artifacts produced	Why it stopped, which is the highest-value field in the schema
`escalation`	trigger, turn index, resolution	Whether the agent can and does decline

Termination must be typed. Published taxonomies distinguish success, maximum turns exceeded, budget exceeded, execution error, and structured-output retry exhaustion.[130] Add two of your own: halted by policy, and aborted by anomaly detection. An organization recording only “the agent finished” has discarded its best diagnostic.

Record the compaction boundary. As the context window fills, older messages are replaced with a summary, and the vendor documentation warns explicitly that “specific instructions from early in the conversation may not be preserved.”[130] If a session compacted and then violated a rule you thought was in force, the boundary event is the explanation.

25.2 Derived Metrics

Termination distribution. The share of sessions ending in each reason, trended. A population that is entirely `success` means the taxonomy is not wired up, not that nothing fails.

Cost as a distribution, never a mean. Runs on the same task can differ by up to thirtyfold in total tokens, so budget against a percentile.[154] Report p50 and p95 cost per task by work class. Two further findings make the mean actively misleading: accuracy often peaks at intermediate cost and saturates above it, and neither the model nor a human expert can forecast a task's cost in advance, with model self-predictions correlating only up to 0.39 and systematically underestimating.[154]

Cost per accepted change. Total spend divided by changes that merged. This is the only cost metric that survives contact with a CFO, because it prices the failures too.

Turn index at termination, split by outcome. Failed trajectories are consistently longer and show higher variance than successful ones.[158] Once you have your own distribution, a long-running session is a signal rather than a curiosity.

Marginal turn cost. Vendor modeling of a fifty-turn session puts input at roughly 5,000 tokens per turn for turns one through ten, 20,000 for turns eleven through thirty, and 35,000 for turns thirty-one through fifty.[159] Cost per turn rises roughly linearly with turn index, which means an agent that is going to fail is cheapest to stop early. Verify the shape against your own sessions rather than adopting the vendor numbers.

25.3 Early-Abort Detection

State this plainly to your team: there is no published detection heuristic with a measured false-positive rate, and no published runaway-loop threshold. What follows is reasoning from measured failure structure, and it should be deployed with instrumentation rather than confidence.

The structure it reasons from is well measured. In failed trajectories the decisive error occurs at a median of step 7 out of roughly 27, the recovery window is a median of one step, observable signals of the failure surface roughly ten steps later, and 82 percent of failures show no termination after the error locks in.[117] The agent keeps working, productively-looking, for the remaining twenty steps. That is where cost is spent after the task is already lost.

Candidate abort signatures, in rough order of confidence:

1. **No diff progress.**
   N consecutive turns with tool calls but no net change to the working tree.

2. **Oscillation.**
   The same file edited back and forth across turns, or the same command reissued with trivial variation.

3. **Rising failure density.**
   The proportion of tool calls returning errors climbing across a moving window.

4 **Denial clustering.**
Repeated policy denials, which indicate an agent probing for a way around a boundary.

5 **Cost slope.**
Spend accelerating without artifact production.

Set thresholds from your own successful-session distribution, abort at a percentile you can defend, and **measure what aborting cost you** by sampling aborted sessions and re-running them to completion. An abort policy without that sampling is a policy that quietly discards good work.

25.4 Two Rules Worth Enforcing in Configuration

Ceilings on every deployment. Maximum turns and maximum spend are separate controls, and at least one major SDK defaults both to unlimited.[130] Default-deny unlimited at the platform level and log every override with an expiry.

Persistent constraints re-injected, not stated once. Anything that must hold for the whole task belongs in a re-injected artifact rather than the opening prompt, because compaction will summarize the opening prompt away. Verify with a test: run a long session, force compaction, and confirm the constraint still binds.

25.5 Failure Signatures

- **Everything succeeds.** The termination field is not populated, or is populated by the harness rather than by outcome.
- **Cost reported as an average.** Hides the thirtyfold spread and produces budgets that are wrong in both directions.
- **Telemetry collected and unread.** Instrument the ratio of telemetry generated to telemetry analyzed and report it. Recording is not detection.
- **Abort thresholds copied from a blog post.** There are no published thresholds. Anything you adopt from outside is somebody's guess about a different codebase.

25.6 What This Rests On

The event schema derives from published termination taxonomies and compaction behavior.[130] The cost findings, the failure-process structure, and the trajectory-length signal are all measured.[154,117,158] The abort signatures and thresholds are this framework's reasoning from that structure, and are explicitly unevidenced.

26 Specifications Agents Can Build From

Produces. A specification template, plus a worked conversion of one real requirement from prose into a form an agent can build against and an oracle can test.

Run this when. Before delegating any work class above A1, and as the standing format for requirements in any team where agents produce implementation.

26.1 Why This Is the Highest-Leverage Chapter

Specification completeness is the second-strongest predictor of delegation success in Section 6.1, and the single largest published improvement attributable to it is stark: one enterprise agent deployment moved from 38.1 to 69 percent success across the cohorts before and after its first preparation changes, "not through better AI models, but through better preparation."[105]

The reverse is measured too, and it is the finding that should change how your organization writes tickets. **Agents cheat far more on ambiguous work.** On unambiguous problems one model hardcoded test values 0.7 percent of the time; on ambiguous problems the rate was 44.4 percent.[185] Ambiguity is not merely inefficient. It is the condition under which an agent stops solving and starts satisfying.

A human engineer resolves an ambiguous requirement by asking, inferring, or noticing a contradiction. An agent resolves it by generating something plausible. That single difference is why the economics of specification have inverted: when implementation was expensive relative to specification, lightweight requirements were rational. They are no longer.

26.2 The Template

Nine fields. Any field you cannot fill is a decision you have not made, and delegating before you make it transfers the decision to a model.

FIELD	CONTENT	WHY IT IS HERE
Intent	One sentence on what changes for whom, and why	The thing an agent cannot infer and will otherwise invent
In scope	Enumerated paths, modules, or surfaces	Bounds the blast radius and the review
Out of scope	What must not change, stated explicitly	Prevents the scope creep that context degradation produces

FIELD	CONTENT	WHY IT IS HERE
Preconditions	System state the change assumes	Removes a class of false premise, the largest single failure mode at 30.7 percent[117]
Acceptance criteria	Machine-checkable, one per line	Drives the oracle; if it cannot drive a check it is not a criterion
Non-functional budgets	Numeric: latency, payload, memory, accessibility conformance, cost	Nothing unstated survives generation at volume
Security classes	Named CWE classes requiring explicit handling	The high-failure classes are known in advance
Oracle reference	Which oracle decides this, per Section 24	Forces the correctness question before work starts
Escalation triggers	Conditions under which the agent must stop and ask	Makes declining a legitimate outcome

Two fields deserve expansion.

Non-functional budgets must be numbers. The population-scale accessibility regression is the clearest evidence of what happens to quality attributes that are not stated and checked: detected conformance failures on 95.9 percent of the top million home pages, up from 94.8 percent, with average errors per page up 10.1 percent in a single year.[37] Nothing downstream will introduce a budget that the specification omitted.

Security classes should be named specifically, because the failure profile is known: generated code passes SQL injection checks at 82 percent and insecure cryptography at 86 percent, against cross-site scripting at 15 percent and log injection at 13 percent.[15] A specification that says “the system shall be secure” transfers the whole problem to a review function that Section 3.3 shows is already saturated. Name the classes that fail.

26.3 Worked: Prose to Buildable

Before. *"Add rate limiting to the public API so we don't get hammered."*

Every failure mode in this chapter is latent in that sentence. Which endpoints? What limit? Per what key? What happens at the limit? Is the limiter shared across instances? What is the behavior when the limiter itself is unavailable? An agent will answer all six, plausibly, and you will discover its answers in production.

After.

> **Intent.** Protect the public read API from a single client exhausting capacity, without degrading service for compliant clients.
>
> **In scope.** `api/public/*` request path; the shared limiter module.
>
> **Out of scope.** Authenticated internal routes; billing; the existing quota system, which is unrelated and must not be touched.
>
> **Preconditions.** Requests carry a resolved client identifier by the time they reach the handler. Redis is available to all instances.
>
> **Acceptance criteria.**
>
> 1. A client exceeding 100 requests in any rolling 60 seconds receives HTTP 429.
>
> 2. The 429 response carries `Retry-After` in seconds and does not include a response body containing request data.
>
> 3. Limits are enforced per client identifier, not per source address.
>
> 4. Counters are shared across instances; two instances observe one budget.
>
> 5. When the limiter backend is unavailable the request is **allowed** and a `limiter.unavailable` metric increments. Fail open, and record it.
>
> 6. Compliant clients at 99 requests per minute observe no added latency above the budget below.
>
> **Non-functional budgets.** Added p95 latency ≤ 5 ms. Memory per instance ≤ 20 MB at 10,000 tracked clients.
>
> **Security classes.** CWE-117 log injection: the client identifier is attacker-influenced and must be encoded before logging. CWE-770 resource exhaustion: the tracking structure must be bounded.
>
> **Oracle.** Authored integration suite `limits/` plus a held-out multi-instance suite the branch does not run. Test files are outside agent write scope.
>
> **Escalation triggers.** If per-client identification is not reliably available at the handler, stop and escalate rather than substituting source address.

Criterion 5 is the one worth studying. Fail-open versus fail-closed is a decision with real consequences, an agent will make it silently if you do not, and its choice will look reasonable in review.

26.4 Specifying Probabilistic Behavior

A requirement for a deterministic component states a behavior. A requirement for a model-backed component cannot, and must instead state a distribution with a threshold, a corpus, and a measurement method.

Not: *the system summarizes documents accurately.*

Instead: *summaries score at or above 0.8 on rubric R for at least 95 percent of evaluation corpus C, measured with a reported confidence interval; below confidence threshold T the system returns a defined refusal rather than a summary; corpus C is refreshed quarterly from production traffic with a rolling holdout.*

Three notes on that shape. Report an interval rather than a point, because a two-point difference on a two-hundred-item set is usually indistinguishable from noise.[74] Refresh the corpus, because any set that has existed publicly for a year should be assumed present in the next model's training data.[77] And specify the failure path, since the common production failure is not an error but a well-formed, confidently wrong answer.

26.5 Instrumentation

- Share of delegated work items carrying a complete template, by work class
- Escalation rate by specification author, which surfaces who writes ambiguity
- Rework attributable to specification defect rather than implementation defect, adjudicated at review
- Agent cheating indicators correlated against specification completeness, which is the local version of the 0.7 to 44.4 percent finding
- Share of acceptance criteria that actually drive an automated check

26.6 Failure Signatures

- **Criteria that no check consumes.** If nothing executes the criterion, it is a wish. Measure the share that drive automated checks and hold it above a floor.
- **The specification written after the code.** Common under delivery pressure, and it inverts the control: the specification then documents what the agent chose rather than constraining it.
- **Escalation triggers absent.** Without a legitimate way to decline, the agent's only path is to produce something, which is precisely the condition the ambiguity finding describes.
- **Quality attributes as adjectives.** "Fast," "accessible," and "secure" are not budgets. They will not survive generation at volume and nothing downstream will supply the number.

26.7 What This Rests On

The preparation effect, the ambiguity-to-cheating link, the accessibility regression, the security class profile, the false-premise failure rate, and the statistical-acceptance and contamination findings are all measured.[105,185,37,15,117,74,77] The nine-field template is this framework's construction. It has not been evaluated against an alternative format, and an organization adopting it should treat the format itself as an experiment with the instrumentation in 27.5 attached.

27 How to Stand Up a Work Class

Produces. A registered work class with an owner, a demonstrated oracle, entry and stop conditions, a funded review budget, and a pilot result that either graduates it or does not.

Run this when. Any time the organization is about to delegate a category of work rather than an individual task. Section 6 gives the taxonomy and the evidence; this chapter is the procedure for putting one into production.

27.1 The Sequence

Step 1. Write the class definition. Name, scope boundary, the change shapes it covers, and the change shapes it explicitly does not. A class defined as “refactoring” is not a class. A class defined as “extract-method refactorings within a single module, no public interface change, no dependency change” is one you can build an oracle for.

Step 2. Demonstrate the oracle before anything else. Run Section 24 to completion, including the immutability test where you attempt a write from a test agent and confirm failure. **A class without a demonstrated oracle cannot exceed A1**, and this is the step organizations skip because the tooling works without it.

Step 3. Set entry conditions. What must be true of a work item before an agent may be dispatched against it. Typically: a complete specification per Section 26, a change-size bound, a path restriction, and a repository whose test suite passes at head. Entry conditions are enforced at dispatch, not documented in a wiki.

Step 4. Choose guards for the known failure mode. Every class has one, and it is usually published. Section 27.3 gives four worked examples. A guard is an automated check specific to the way this class goes wrong, and it runs in addition to the oracle.

Step 5. Model the review cost and fund it. Estimate reviewer hours per accepted change, multiply by projected volume, and compare against available senior engineering hours. If the answer exceeds capacity, the class does not launch at that volume. Every published deployment at scale independently discovered that review throughput rather than agent capability is the binding constraint, and one report measured 16.5 review comments per merged agent pull request against 12.4 for human ones.[105]

Step 6. Define stop conditions that halt. Not alert. Halt. Typical set: merge rate below a floor, revert rate above a ceiling, review backlog beyond N days, oracle modification attempt detected, visible-to-held-out gap beyond threshold, cost per accepted change beyond budget.

Step 7. Pilot with a termination decision scheduled. Two to four weeks, one team, bounded repositories, full instrumentation, and a date on which someone decides to graduate, extend, or kill it. A pilot without a scheduled decision becomes production by default.

Step 8. Set the baseline before you compare. Measure human merge rate, revert rate, and review cost in the same repositories first. Agent performance is meaningless without it: one enterprise report puts agent merge rate at 67.9 percent against 87.1 percent for employees and 79.7 percent for community contributors **in the same repository**.[105] Without that denominator, 67.9 percent reads as either impressive or alarming depending on the reader’s priors.

Step 9. Graduate on evidence, at a named authority. Section 19.4 is the gate. Graduation raises the tier, widens scope, or both, and it requires operating evidence rather than absence of complaint.

27.2 Setting Expectations Before the Pilot

Two variables explain more outcome variance than tool choice, and both are knowable in advance. Set expectations from them or the pilot will be read as a verdict on the tool.

Language. One enterprise test-generation system reports viable-test rates of roughly 80 percent for Python, 40 percent for Go, and 20 percent for Java, on the same tool in the same monorepo.[123] Security pass rates for generated code show the same shape at 62 percent for Python against 29 percent for Java.[15]

Codebase age. Greenfield beats brownfield measurably, at 77.3 percent against 67.9 percent in one organization, and an independent evaluation of an autonomous agent scored zero for four on modifying existing projects.[105,122]

Write the expected range into the pilot plan before it runs. A Java brownfield pilot that lands at 45 percent is performing about as the evidence predicts; the same number in a Python greenfield repository is a problem.

27.3 Guards by Class

CLASS	KNOWN FAILURE MODE	GUARD
Test generation	Tests that pass but detect nothing	Mutation-score floor on the generated suite, not a coverage floor.[170]
Flaky-test repair	The flake disappears because assertions were removed	Assertion-count delta must be zero or positive; semantic diff on the test body; mutation check that the repaired test still kills its original mutant. The rejection analysis of a deployed system names altered test semantics and removed assertions among rejected fixes.[161]
Dependency upgrade	Model-chosen version carries a known CVE	Versions resolved by policy against a vulnerability database rather than accepted as specified; 36.7 to 55.7 percent of tasks in one study carried a known CVE in the model-specified version.[64]

CLASS	KNOWN FAILURE MODE	GUARD
Security remediation	The alert closes without the vulnerability being fixed	Independent verification by something other than the alert: a functional test plus a security test the fixing agent never saw. The best vendor benchmark still fails roughly one in six on secure-and-functional first try, and an independent user study found 75 percent of proposed fixes unsuitable to apply as-is.[163,164]
Migration	Semantic drift the suite does not catch	Differential comparison against the pre-migration behavior; shard into independently revertible changes rather than one atomic change.[167]
Documentation	Confidently wrong prose	Executable examples, reference validation, and signature comparison. Absent those, A1 and human review.

27.4 Worked: Standing Up Flaky-Test Repair

Definition. Repair of tests identified as flaky by the existing detection pipeline, in Go, within `services/*`, excluding tests touching payment paths.

Oracle. Reproduction of the flake under repeated execution, then repeated passes after repair, at a defined run count. Independent because the reproduction harness is platform-owned. Immutable because test files are outside agent write scope for everything except the target test, and the target test's assertion count is checked separately.

Entry conditions. The flake is reproducible by the harness; the test is under fifty lines; the owning team has opted in.

Guards. Assertion count must not decrease. A semantic diff flags any change to assertion expressions for mandatory human review. The repaired test must still fail against a mutant that the original test killed.

Review cost. From the published deployment, plan against roughly 17.7 percent end-to-end yield: of 1,115 reported flaky tests, 71.6 percent reproduced, fixes were produced for 47.6 percent of those, and 51.8 percent of generated fixes were accepted and landed.[161] **Budget review against the fixes produced, not the fixes accepted.** A team planning against 51.8 percent will under-provision review by roughly a factor of three.

Stop conditions. Acceptance below 35 percent over two weeks. Any assertion-removal guard trip. Reviewer backlog above five working days.

Pilot. Four weeks, one team, decision date on the calendar, baseline measured first.

27.5 Instrumentation

- Merge, revert, and defect-escape rate by class, against the per-language and per-repository baseline from step 8
- Declared review cost against measured review hours
- Guard trip rate by guard, with every trip reviewed
- Cost per accepted change
- Classes throttled or tier-reduced on evidence in the trailing year, where zero indicates a register rather than a control

27.6 Failure Signatures

- **The class that has never been throttled.** It will saturate review eventually. A register in which nothing has ever been reduced is a list.
- **Acceptance rate quoted as yield.** The single most common planning error in this literature. Plan the funnel.
- **Pilot without a baseline.** Produces a number nobody can interpret, and the interpretation defaults to whoever is most invested.
- **Guards deferred to phase two.** The failure mode each guard addresses is published in advance. Launching without them is choosing to discover a known problem in production.

27.7 What This Rests On

The review-cost constraint, the language and codebase-age effects, the per-class failure modes, and the flaky-test funnel are all measured.[105,123,15,122,170,161,64,163,164,167] The nine-step sequence and the pilot structure are this framework's construction.

28 The Context Substrate in Practice

Produces. An entitlement-preserving retrieval layer, an indexed catalog of code and ownership, and a short, evaluated set of instruction files.

Run this when. Early. Section 8.1 identifies this as the platform layer with the largest measured effect, and it is the one most often deferred in favor of buying tools.

28.1 Build the Substrate, Not the Markdown

The evidence here is genuinely contested and it points away from where most organizations put their effort.

Instruction files improve efficiency and probably not success. A controlled study across multiple models and agents, using both generated and developer-written context files, found that "providing context files does not generally improve task success rates, while increasing inference cost by over 20% on average."[131] A paired within-task study of 124 pull requests found median wall-clock time down 28.6 percent and median output tokens down 16.6 percent, while explicitly not evaluating task success.[132] A field study of 15,549 agentic pull requests across 148 projects found near-symmetric outcomes: 27.7 percent of projects raised merge rate by at least 20 percent after introducing an instruction file, and 26.4 percent lowered it by at least 20 percent.[133]

Retrieval is where the measurable failure is. Across 427 samples from 25 repositories, interactive agents never accessed the relevant files on 27 to 35 percent of samples despite exploration, and the median relevant evidence occupies only 4.7 percent of the file containing it.[203] Context acquisition is a distinct failure surface from patch generation, and it is the one with room to move.

So: index the code, expose the catalog, preserve entitlements, and keep the markdown short.

28.2 Entitlement-Preserving Retrieval

This is the single property to copy first, and one published enterprise implementation states it directly: internal tool-protocol servers front the wiki, product management system, data warehouse, CRM, and workspace, **preserving existing user permissions so agents retrieve only what the invoking human can already see.**[204]

The procedure is short and the verification is the part that matters.

1. Resolve the invoking human's identity at the start of every session and carry it as the retrieval principal.
2. Enforce entitlement at the data layer, not by filtering results after retrieval. Filtering after means the content transited the model before being removed.

3 Scope the index itself by classification, so that regulated data requires a separate, separately authorized path.

4 **Verify adversarially.**
Instruct a test agent to retrieve something the invoking human cannot see and confirm refusal. A policy statement is not this step.

28.3 What to Index

In descending order of measured or plausible value:

Code, semantically. The index earns its keep as repository size grows: the vendor advocating indexing reports code retention improving 0.3 percent overall but 2.6 percent on codebases over a thousand files.[137] Two major vendors hold publicly opposite positions here, each publishing evidence favoring its own architecture, so the honest reading is scale-dependence rather than a settled answer.[136,137]

The service catalog and ownership topology. One published implementation injects component architecture, dependency graphs, ownership topology, and architectural decisions into agent sessions from its internal developer portal, with session transcripts flowing back into the same portal, making the catalog both the context source and the audit surface.[195]

Architecture decisions. ADRs as retrievable context are the mechanism by which an agent can know why the codebase is shaped as it is. This is reasoning rather than measurement; no study has isolated the effect.

A component catalog answering "does something like this already exist." This is the most useful counter to the duplication trend in Section 3.6, where refactoring fell from 21 percent of changed lines to 3.8 percent while copy-paste rose from 9.4 to 15.7 percent.[38] Nothing in a generative workflow rewards finding the existing implementation.

A knowledge graph, if you are large enough. One organization operates a context graph of roughly 40 million entries across about 150 node and edge types, explicitly built to reduce tokens, turns, and latency.[103] That is evidenced by essentially one company via a conference talk. Treat it as promising and single-sourced.

28.4 Instruction Files: Short, Specific, Evaluated

Given the null results, the defensible posture is narrow.

Include non-obvious house rules an agent cannot infer, build and test invocation, conventions that differ from ecosystem defaults, and **security content**, which is almost always missing: a study of 2,303 context files found testing content in 75 percent, implementation detail in 69.9 percent, and architecture in 67.7 percent, with security and performance each appearing in only 14.5 percent.[134] Published guidance exists on what security content belongs there, including preferring well-vetted libraries, using official package managers, pinning versions, and verifying integrity by checksum or signature.[99]

Exclude repository overviews, which are the part vendors most commonly recommend generating and the part with the least support.[131]

Evaluate rather than generate. Before rolling a file out, run the same corpus of tasks with and without it and compare success, cost, and review rework. Projects that improved had files with a median of 976 words against 569 for projects that declined, so length and structure matter, but your own measurement matters more than either number.[133]

Review them as code, because they are an execution surface. Across the disclosed vulnerabilities in agentic development tooling, the vulnerable object is repeatedly the configuration file rather than the code.[19,20,21,22]

28.5 Instrumentation

- Retrieval coverage: share of repositories indexed, and share of sessions where the agent reached the relevant file
- Adversarial retrieval test result, refreshed on every substrate change
- Instruction file effect: paired task success and cost, per file, before and after
- Duplication and reuse trend, as the outcome measure for the component catalog
- Median tokens per session, which is where substrate improvements show up first

28.6 Failure Signatures

- **Retrieval scoped by prompt.** "Only look at what the user can see" is an instruction. Entitlement belongs at the data layer.
- **Instruction files that grow.** They accumulate because adding a line is easier than diagnosing a failure. Cap the length, review on change, and re-evaluate periodically.
- **Generated repository overviews.** The specific artifact the evidence does not support, and the one most commonly auto-produced.
- **The catalog nobody queries.** If agent sessions do not reach it, it is documentation with an API.

28.7 What This Rests On

The context-file null results, the efficiency effect, the field outcome distribution, the retrieval failure rate, the content-composition study, and the indexing scale effect are all measured, though several are preprints and two are vendor-published with opposing conclusions.[131,132,133,203,134,136,137] The enterprise implementations are self-reported, and one is sourced through a conference talk rather than a first-party engineering publication.[204,195,103] The indexing priority order is this framework's judgment.

29 The Review Operating Model

Produces. A risk-tiering rule set, an auto-merge policy with a measured false-negative rate, a reviewer rotation, and a seeded-defect probe on a schedule.

Run this when. As soon as agent-authored change is a material share of volume. Section 3.3 shows review saturation arriving before organizations notice it, and the merge record looks identical whether review happened or was waived.

29.1 The Problem, Stated Numerically

Telemetry across roughly 22,000 developers found median review time up 500 percent alongside pull requests merged without review up 31.3 percent.[119] Those two together are the signature: reviewers are queuing and waiving simultaneously. A study of merged agentic pull requests found 79.1 percent of a manually inspected sample showing no observable reviewer interaction at all.[31]

Uniform review does not survive this. The question is not whether to triage but on what basis.

29.2 Risk-Tier the Change, Not the Author

Reviewers already do this implicitly, taking a median 3.92 hours on security-relevant agent pull requests against 0.11 hours otherwise.[171] Formalize what is already happening.

TIER	TRIGGERS	REQUIREMENT
R3 — **Senior mandatory**	Authentication, authorization, cryptography, payment paths, data deletion, migrations, public interfaces, infrastructure definitions, agent and tool configuration, CI configuration, dependency manifests	Named senior reviewer; no auto-merge under any circumstance; second reviewer for R3 changes that are also agent-authored
R2 — **Standard review**	Business logic, internal interfaces, non-trivial refactoring	Any qualified reviewer; oracle must pass; auto-merge prohibited
R1 — **Light review**	Mechanical change within a registered work class, bounded size, oracle strong and passing	Sampled human review at a defined rate; auto-merge permitted under 30.3

Tiering is enforced by path and change-shape rules in the platform, not by reviewer judgment at the moment of review. Configuration files belong in R3 and are the most commonly misfiled, because they do not look like code.

29.3 Auto-Merge, Honestly

Auto-merge is the practice that makes Stage 3 economically different from Stage 2, and it is also where an organization can quietly stop verifying. One published enterprise practice describes it as "auto-merging what's safe, concentrating review where it matters most," and that is the entire published record.[101]

Four preconditions, all required.

1. The change is R1 inside a registered work class with a demonstrated, immutable oracle.
2. The oracle has a **measured** false-negative rate from Section 24 step 6. Not an assumed one.
3. A held-out signal runs post-merge, with the visible-to-held-out gap monitored.
4. Sampled human review continues at a rate high enough to detect drift, with the sample reviewed by a senior engineer and the findings fed back.

Set the sampling rate from your own defect economics, not from a published figure, because there is none. **No published study measures human review effectiveness on agent-authored code against a ground-truth defect set, and none measures rubber-stamping rates in agent pull request review.** Any auto-merge policy is therefore built on your own measurement or on nothing.

29.4 Collaborate on Hard Changes

The most actionable review finding in the enterprise literature is not about review at all. When humans committed directly into an agent's pull request rather than reviewing and returning it, success rose from 55 to 86 percent.[105]

That is a large effect and it argues against a pure reviewer posture on complex work. The operating rule: for R2 and R3 changes that fail once, the second attempt is collaborative rather than another review cycle. Reviewing and returning is efficient for mechanical defects and expensive for design ones.

29.5 The Seeded-Defect Probe

Automation complacency is the same underlying phenomenon as automation bias, with attention allocation at its center rather than two separate failure modes.[210] A reviewer whose attention has been reallocated by reliable automation does not become a worse reviewer; they become a slower one to notice.

The probe is the only mechanism in this framework that measures whether review is actually happening.

1. Construct defects representative of what your oracles do not catch: authorization boundary errors, output encoding in the wrong context, silent fallbacks, off-by-one in pagination, error suppression.

2 Seed them into the ordinary review stream at a low rate, indistinguishable from real changes.

3 Record catch rate, median review time, and diff size at approval for both seeded and unseeded changes.

4 Report catch rate as a control-efficacy metric, and treat a decline as a signal to reduce merge rate rather than to exhort the team.

5 **Tell the team the probe exists**
before you run it. Announce the practice, not the instances. A probe run covertly is an evaluation of individuals; a probe run openly is an evaluation of the system, and only the second produces honest behavior.

Rotate what reviewers see. Attention settles on the predictable, so a reviewer who always reviews the same work class will detect less within it over time. This is a reasoned application of the complacency literature to review scheduling and it is not measured in a software setting.

29.6 Push the Mechanical Down

Human review is the most expensive layer and the evidence on what it delivers is less flattering than most organizations assume: a classification of 570 review comments found code improvements at 29 percent and defect finding at only 14 percent, with the remainder understanding, knowledge transfer, and design discussion.[181]

Meanwhile up to 76 percent of what review catches is in principle detectable by static analysis, against roughly 25 percent that generic tools actually deliver, with the shortfall being project-specific rules no generic tool can express.[180] That gap is the work queue. Every house rule you codify as machine-checkable lint is review capacity returned.

Use human review for design, architecture, intent, and whether the change should exist at all. Those are the things it is measurably good at and the things nothing else does.

29.7 Instrumentation

- Absorption ratio: review hours required against senior engineering hours available
- Share of changes merged without human review, by tier and by authorship class
- Median and 95th-percentile review duration, and diff size at approval
- Seeded-defect catch rate and median review time on seeded changes
- Auto-merge false-negative rate from post-merge held-out results
- Share of house rules codified as automated checks

29.8 Failure Signatures

- **Approval rate near 100 percent with review times in seconds.** Not a control. Instrument both and act on the pair.
- **Auto-merge scope creeping.** It expands one work class at a time, each individually defensible. Require the four preconditions on every expansion.
- **The probe that always passes.** Either the seeded defects are too easy or they are recognizable. Rotate their shape.
- **Rising review exceptions.** Every exception granted to hit a date is a permanent hole unless it carries an expiry.
- **Configuration filed as R1.** The most common tiering error, and the one every major disclosed vulnerability in agentic tooling went through.

29.9 What This Rests On

The saturation telemetry, the no-interaction merge finding, the security review-latency differential, the collaboration effect, the review-efficacy classification, the static-analysis ceiling, and the automation-complacency synthesis are measured.[119,31,171,105,181,180,210] The auto-merge preconditions, the probe design, and the reviewer rotation are this framework's construction, and the absence of any published measurement of review effectiveness against agent-authored code means every organization is calibrating this on its own data.

30 Qualifying an Agent for Write Access

Produces. A qualification record for one agent deployment: what it was tested against, what it scored, who approved it, at what tier, and when it must re-qualify.

Run this when. Before any deployment reaches A2, on every model version change, and on every scope expansion.

30.1 Start by Discarding the Benchmark

Benchmark scores are not a qualification signal, and using them as one is the most common error in this decision. The gap is measured from three directions.

Agents resolving 72.8 percent of a widely used verified benchmark resolve 18.75 to 25 percent of realistic multi-file evolution tasks drawn from release notes.[115] One major provider retired that benchmark in February 2026, having audited 138 problems and found 59.4 percent with material test-design issues, alongside contamination evidence.[124] And an audit of 731 successful trajectories found 63 percent of resolutions retrieved rather than derived, with 57 percent locating the merged upstream fix on the public web and 9 percent mining bundled git history for the future fix commit.[125]

Benchmarks measure capability on curated tasks under no organizational constraint. Qualification measures behavior under yours.

There is no published enterprise standard for agent qualification, no accepted protocol, and no agreed criterion for granting repository write access. What follows is a defensible internal protocol. Adopt it, instrument it, and describe it internally as invented rather than inherited.

30.2 The Four Stages

Stage A — Sandbox. The agent runs in an isolated environment with no access to production credentials, no network egress beyond an allowlist, and a disposable copy of a real repository. Objective: establish that it can complete representative tasks at all, and observe how it fails. Run with network and git history restricted, so that what you measure is engineering rather than retrieval.

Stage B — Shadow. The agent receives real tasks from the real queue and produces real diffs that are **scored and discarded**. Humans do the work independently. You compare.

This is the highest-information stage and the one to invest in. It is also, honestly, the least evidenced: the pattern follows directly from a well-documented pre-AI practice in which a control path returns to callers while a candidate path runs alongside and mismatches are recorded,[184] and **no published enterprise deployment of shadow mode for agent output exists.** Build it with instrumentation and treat your own numbers as the evidence base.

What to score in shadow:

- Would this diff have merged, adjudicated blind by a reviewer who does not know the author
- Diff distance from what the human actually did, and whether differences are stylistic or semantic
- Oracle result, including the held-out signal
- Scope adherence: did it touch only declared paths
- Cost per task at p50 and p95
- Failure shape, categorized against Section 5.8

Stage C — Canary. Write access to a low-blast-radius repository, R1 changes only, every change reviewed, tier A2. Objective: observe real merge behavior, real review cost, and real conflict rate with human work in flight.

Stage D — Graduated scope. Expansion by repository, work class, or tier, one dimension at a time, each with its own evidence. Never two at once, because a regression after a double expansion is unattributable.

30.3 Criteria That Are Organizational, Not Technical

Set thresholds from your own baseline per Section 27 step 8. The criteria below are the dimensions; the numbers are yours.

DIMENSION	WHAT IT MEASURES	WHY IT MATTERS
Merge rate against the human baseline in the same repository	Capability under your constraints	The only comparison that means anything
Revert rate	Whether merged work survives	The least-measured dimension in the literature; one report gives 0.6 percent for agent changes against 0.8 percent for others[105]
Review cost per accepted change	The load it imposes	16.5 comments per merged agent change against 12.4 for human ones in one deployment[105]
Scope adherence	Whether it stays inside its envelope	Predicts blast radius under expansion
Conflict rate	Behavior alongside other work	19.8 percent within one agent product, 41.7 percent across products[32]
Oracle interaction	Attempts to modify tests, configuration, or grading	Any attempt is a qualification failure, not a data point

DIMENSION	WHAT IT MEASURES	WHY IT MATTERS
Escalation behavior	Whether it declines when it should	An agent that never escalates has learned that producing scores better than declining
Visible-to-held-out gap	Optimization toward the visible check	The reward-hacking indicator from Section 24

Two of these are pass-fail rather than threshold: an oracle modification attempt, and a zero escalation rate across a meaningful sample.

30.4 Re-Qualification

On model version change, always. A provider model change is a production change to the system whether or not your code changed. Behavior can shift materially under a stable API surface: one measurement found accuracy on a specific task falling from 84 to 51 percent across three months, alongside reduced instruction-following.[87]

On scope expansion, always. Verification degrades with scope rather than with model quality, and the reward-hacking gap grows roughly 27 percentage points per tenfold increase in code size.[114] A qualification earned at module scale does not transfer to service scale.

On a calendar, at minimum quarterly. Drift accumulates in the substrate, the codebase, and the model together.

30.5 Instrumentation

- Qualification records with approver, criteria, scores, and expiry, for every A2-and-above deployment
- Share of deployments operating on an expired qualification, target zero
- Shadow-mode agreement rate over time, which is your leading indicator of capability change
- Re-qualification pass rate on model version change

30.6 Failure Signatures

- **A benchmark score in the qualification record.** It measures a different thing under different conditions.
- **Qualification once, forever.** The model changed underneath it.
- **Shadow mode skipped for time.** It is the only stage that produces evidence without risk, and it is always the one cut.
- **Expansion on two dimensions at once.** Any regression becomes unattributable, and the response becomes a guess.
- **Scores from an environment with network and full git history.** You measured retrieval.

30.7 What This Rests On

The benchmark-transfer gap, the benchmark retirement audit, the retrieval finding, the conflict rates, the revert and review-cost figures, the model drift measurement, and the scope-scaling result are measured.[115,124,125,32,105,87,114] The four-stage protocol, the criteria set, and the pass-fail rules are this framework's construction, and shadow mode specifically has no published enterprise precedent for agent output.

31 Fleet Operations

Produces. Cost attribution per agent and per task, enforced quotas, tested kill switches, and a telemetry pipeline whose analyzed fraction is known.

Run this when. Once agents outnumber the people who can watch them individually, which arrives earlier than expected.

31.1 Cost Is a Reliability Signal

Runaway consumption is the earliest observable indicator of an agent in a loop, an agent being manipulated, or an agent operating outside its intended scope. Treating cost as a finance concern rather than an operational one loses that signal.

The unit is cost per task, not cost per token. Agentic workloads consume roughly a thousand times more tokens than conversational use, so falling unit prices coexist with rising bills.[154] Per-token price declines have been rapid, at roughly fortyfold per year for a fixed capability threshold, but with a spread of ninefold to nine-hundredfold depending on task, and that analysis dates from early 2025.[45]

Attribute below the account. Team, repository, work class, agent deployment, and task. Observed total cost per engineer runs $200 to $600 per month including tokens against seat list prices of roughly $19 to $60, a four- to ten-fold multiple on the procurement line item, with governance infrastructure a further $50,000 to $250,000 annually.[40]

Ceilings that halt. One organization caps every employee at $1,500 in monthly token spend per AI coding tool; at two tools that is roughly $36,000 per engineer per year.[196] Another alerts on individual daily spend above a threshold.[204] Both are simple, and both work because they are enforced rather than reported.

Route everything through a gateway. One published implementation runs roughly 100 million model requests per day through a governance layer holding a sub-100-millisecond latency budget.[103] Centralization is what makes attribution, quota, and anomaly detection possible at all.

31.2 Where the Savings Actually Are

Model routing research is strong in benchmark settings, reporting up to 84 percent cost savings with competitive accuracy and 97 percent of a frontier model's quality at 24 percent of cost under time constraints.[199] **No enterprise has published measured savings from production model routing or caching.**

The one large published fleet-level saving came from somewhere else entirely: response shaping at the tool gateway, reported as a greater than 40 percent reduction in token usage across the agent fleet.[103] Compact tool responses, trimmed schemas, and on-the-fly result summarization at the gateway are a different and better-evidenced lever than model selection, and almost nobody discusses them.

Start with response shaping. Treat routing as an experiment.

31.3 Kill Switches That Reach

Three scopes, each tested, each with a measured time-to-effect: one agent, one agent class, the whole estate.

The gap to test for is reach. Kill switches exist in the orchestrator and typically do not reach agents embedded in SaaS platforms, agents in partner environments, or agents running on a developer's machine against corporate credentials. Run the drill against one agent in a third-party platform specifically, because that is the case that fails.

Pair the switch with credential revocation, and measure that separately. Halting the process while its tokens remain valid is a partial control.

31.4 Telemetry You Actually Read

Emit in a versioned schema and expect to re-instrument. All generative-AI semantic convention attributes, spans, metrics, and events currently carry development status with none stable, and the conventions moved to a dedicated repository in June 2026 with no tagged release.[90] Pin the version, treat attribute names as a contract with your own pipeline, and do it anyway.

Instrument the generated-to-analyzed ratio and report it. Recording is not detection, and the ratio degrades silently as volume rises. This single metric is the difference between an evidentiary asset and a control.

Two published instrumentation choices are worth borrowing. **Provider failures** as a first-class signal, which has no analogue in conventional monitoring.[202] And **session transcripts flowing back into the service catalog** after each session, which is architecturally distinct from span-based tracing and gives organization-wide visibility into what agents did and for whom.[195]

No enterprise has published an agent-fleet observability architecture — no data on trace volume, retention, storage cost, sampling strategy, or how session traces join to code outcomes. You are designing this from first principles.

31.5 Concurrency

Bound concurrent agent work streams per repository and partition scope by path. The measured cost of not doing so is a textual conflict rate of 19.8 percent within one agent product, rising to 41.7 percent when pull requests from different agent products overlap, across 33,596 agent-authored pull requests of which 79.4 percent temporally overlapped with others.[32]

The multi-vendor implication is a procurement input no vendor comparison will surface: running two agent products against one repository roughly doubles the conflict rate. If the organization wants product diversity, partition by repository rather than mixing within one.

31.6 Instrumentation

- Cost per task at p50 and p95, and cost per accepted change, by work class and team
- Spend against quota by team, with anomaly alerts and enforced halts
- Kill-switch time-to-effect by scope, from live test, including one third-party platform
- Generated-to-analyzed telemetry ratio
- Conflict rate, with cross-product concurrency broken out
- Share of tool access flowing through the governed gateway rather than point integrations

31.7 Failure Signatures

- **Cost visible only at the account level.** Cannot detect a looping agent, support a chargeback conversation, or inform a tier decision.
- **A kill switch that has never been fired.** Untested is a documentation artifact.
- **Telemetry volume as a success metric.** Volume is cost. The analyzed fraction is the control.
- **Routing adopted for savings on benchmark evidence.** No production measurement exists. Response shaping does have one.
- **Point integrations proliferating.** Every tool server stood up by a team to make an agent useful is an ungoverned credentialed path into the estate.

31.8 What This Rests On

The token-consumption profile, the price-decline range, the per-developer cost figures, the spend cap, the gateway throughput, the response-shaping saving, the routing benchmark results, the telemetry convention status, and the conflict rates are all published, though several are vendor-reported or third-party summaries of conference talks.[154,45,40,196,204,103,199,90,202,195,32] The kill-switch drill design and the concurrency partitioning rule are this framework's construction.

32 The First Ninety Days, Worked

Produces. A sequenced ninety-day program with named artifacts, so that adoption becomes a set of deliverables rather than a list of intentions. Section 33 carries the sequence forward to month eighteen.

Run this when. Starting. This chapter assumes an organization somewhere between Stage 1 and Stage 2 with undeclared pockets of Stage 3, which is where most are.

32.1 Weeks 1–2: Find Out What Is Actually Running

You are measuring the present, not planning the future. Four artifacts.

The population inventory. Not what was procured. What is running. Pull from the identity provider, cloud control planes, repository integrations, developer environments, SaaS admin surfaces, tool-protocol servers, and direct model API calls originating in code and CI. Label the coverage gaps explicitly on the document itself, because the number will be read as complete.

The stage assessment. For every repository with agent participation, answer three questions: can agents self-approve, can they trigger CI without human authorization, can they push to protected branches. Then enumerate every configured bypass actor and the date each was added. Undeclared Stage 3 lives in that list.

The credential exposure result. Compromise a test agent execution context and enumerate every credential obtainable and its remaining validity. One afternoon. It usually reframes the program more than anything else in this chapter.

The oracle map. For every category of work already delegated, name the mechanism that decides correctness and whether the agent can modify it. Most organizations discover at least one work class running with no oracle at all, and that discovery is the finding that justifies the next ten weeks.

32.2 Weeks 3–4: Instrument Before Changing

Changing anything before this point means you cannot tell whether it helped.

Loop telemetry live, per Section 25: the six events, typed termination, cost as a distribution.

The absorption ratio measured, per Section 29: review hours required against senior engineering hours available. Split change volume by authorship class if the data permits, and if it does not, record that as a provenance finding rather than an inconvenience.

Baselines captured, per Section 27 step 8: human merge rate, revert rate, and review cost per repository, per language. Every later comparison depends on this and it cannot be reconstructed retroactively.

The consumption baseline, per Section 31: actual spend per team, license and consumption separated, against budget.

32.3 Weeks 5–8: Close the Boundary and Build One Oracle

Two workstreams in parallel, one security-shaped and one verification-shaped.

Close the merge boundary. No self-approval, no self-authorized CI execution, code-owner review on configuration and dependency manifests, concealed-character scanning on ingested content. **Verify by demonstration**, not documentation: attempt each and confirm failure. Then implement the four hardened-boundary interventions from Section 7.4, which is the highest measured return available in this framework at an 87.7 percent relative reduction in exploit rate with task success essentially unchanged.[113]

Give every A2-and-above agent a unique identity and a named accountable human, and eliminate agents operating under human credentials.

Build one oracle end to end, per Section 24, for the work class with the strongest published evidence. Do not start with documentation. Take it through all seven steps including the immutability test and the calibration run, because the point of this workstream is that the organization learns the procedure on a case where it works.

Set turn and budget ceilings on every deployment and default-deny unlimited at the platform level.

32.4 Weeks 9–12: Run One Work Class Properly

One class, one team, full instrumentation, a scheduled decision date.

Register it per Section 27 with an owner, tier, oracle, entry conditions, guards, and stop conditions. Model the review cost and confirm the capacity exists before launch. Run the pilot against the baseline captured in weeks 3 and 4.

In parallel, three things that do not depend on the pilot:

- **Instrument review integrity** — approval rate, review duration, diff size at approval—and report it.
- **Run the first toolchain red team**: injection through an issue and a pull request description, a non-existent dependency, a configuration file with concealed characters, an attempted self-approval, and an attempted write to a test file.
- **Announce and run the first seeded-defect probe**, per Section 29.5.

32.5 What Ninety Days Does Not Buy

Stage 2 entry requires Level 2 control maturity across all fifteen dimensions in Section 18. Ninety days at this pace typically produces Level 2 on identity, the merge boundary, loop instrumentation, and one work class, with the platform dimensions trailing.

That is the correct outcome, and it should be reported as such. The failure mode is declaring Stage 2 because the visible work is done. The context substrate in Section 28, the evaluation infrastructure in Section 8.5, and provenance verification in Section 15 are quarter-two work, and Stage 3 requires two quarters of operating evidence on top of that.

Anyone promising an AI-native engineering organization in a quarter is selling the tooling, not the operating model.

32.6 The Sequencing Trap

The temptation is to begin with policy, because policy is fast to write and visible to leadership. Policy written before the ground truth of weeks 1 and 2 governs a population that does not exist and misses the one that does.

The second temptation is to begin with a tool selection. Section 21 exists for that decision, and Section 8 is the reason it is not the first one: the published outcome differences between organizations are platform differences rather than tool differences.

32.7 What This Rests On

The hardened-boundary result is measured.[113] The sequence, the artifact set, and the ninety-day scope estimate are this framework's construction, derived from the entry criteria in Section 4.2 rather than from any published implementation timeline. No organization has published a staged adoption timeline with outcome data against it.

33 Beyond Ninety Days

Produces. An operating plan for months four through eighteen: what to build once the boundary is closed and one work class is running, and what evidence Stage 3 requires that cannot be manufactured retroactively.

Run this when. The ninety-day program in Section 32 has produced its artifacts and the first work class has a decision date behind it. Section 4.2 gates Stage 3 on two quarters of operating evidence, which means the clock on this chapter starts the day the first class goes live, not the day someone decides to pursue Stage 3.

33.1 Months Four Through Six: Operate and Instrument

The first ninety days close the merge boundary and prove one oracle. This quarter is where the platform work that makes the second, third, and fourth work classes cheap actually happens.

Replace standing credentials with ephemeral issuance. Task-scoped, audience-scoped, carrying an actor chain that survives the hop. Long-lived tokens on developer machines are what turned a compromised dependency into a thousand-repository credential leak in Section 40, and the remediation the affected vendor shipped was exactly this substitution.

Move architectural constraints out of documentation and into blocking checks, including a duplication threshold. House rules that live in a style guide govern nobody; house rules expressed as machine-checkable lint scale to any volume. This is also the cheapest available answer to the failure mode in Section 35, where roughly a fifth of rejected fixes were rejected for not using the organization's own helper APIs—conventions the agent had no way to learn.

Pilot diff-scoped mutation testing on agent-authored modules. Manage the unproductive-mutant rate before gating on it. Section 37 is the evidence: mutation-guided generation killed mutants at roughly six times the rate of its coverage-guided predecessor, and half its useful tests added no line coverage at all.

Add a held-out verification signal for every work class above tier A2, and begin trending the gap between visible-suite and held-out-suite results. Almost no organization computes this, and it is the only early indicator of optimization toward the visible check.

Restrict network access and git history during acceptance and qualification, so that acceptance measures engineering rather than retrieval.

Emit and verify provenance. Signed artifacts, release manifests that distinguish agent-authored components, SBOMs covering AI components. Verification at consumption, not merely production.

Route model and tool calls through governed gateways with per-team quotas and cost attributed per agent and per task. Cost attribution is the item most often deferred and the one that is hardest to add later, because it requires the call path to have been instrumented from the beginning.

Test separation of duties by attempting to violate it, quarterly, and record the result as evidence rather than as an assertion.

Test kill switches with a measured time-to-effect, including at least one agent running inside a third-party platform where the switch is somebody else's to throw.

Begin the seeded-defect probe on review and measure the catch rate. No published study measures human review effectiveness against agent-authored code, so this number will not come from the literature and has to be produced internally.

33.2 Months Seven Through Eighteen: Qualify for Stage 3

Two quarters of trended operating evidence per work class. Merge rate, revert rate, defect escape, review cost, conflict rate, and incident involvement. This is the requirement that cannot be compressed, and attempting to compress it is the most common way organizations arrive at Stage 3 by accretion rather than by decision.

Continuous validation against production traffic, alerting on distribution shift rather than on error rate alone.

An auto-merge policy defined by change class, with a measured false-negative rate. Not a belief about which changes are safe. Section 38 shows that a temporal control—cohort gating, working-hours constraints, and post-deployment failure correlation—can substitute for per-change verification at sufficient scale and sufficiently low blast radius per change. It is a legitimate design and it is not the same thing as verification. An organization adopting it should say which one it is buying, and should publish internally the failure rate that the case study's source does not.

Fleet-level identity, cost attribution, and observability, with the generated-to-analyzed telemetry ratio instrumented. Telemetry that is emitted and never read is a cost center wearing the costume of a control.

Independent assurance against this framework's evidence requirements rather than self-assessment. The distinction that matters is whether someone outside the delivery organization can reproduce the evidence.

Funded consolidation work driven by measured duplication and reuse trends. Generation is elastic and consolidation is not; without a standing budget for it, duplication accumulates at the rate agents produce it.

Deliberate management of the junior pipeline and of comprehension-preserving usage patterns, with the instrumentation to know whether it is working. Nobody has published that answer, so the organization will have to produce its own.

33.3 What This Period Does Not Buy

It does not buy a defect-escape rate that can be compared against anyone else's, because no organization publishes one—a gap Section 42.6 names directly. Internal trend is the only comparison available, which is a further reason the baselines from Section 32.2 matter more than they appear to at the time.

It does not buy a cost model with external reference points. Of the eight cases in Part VI, one explicitly excludes compute and continuous-integration cost from its analysis, one reports wall-clock and no monetary figure, and the rest report nothing at all.

And it does not buy Stage 3 by elapsed time. Two quarters of evidence is a floor, not a schedule.

33.4 Failure Signatures

- **Evidence assembled after the decision.** Metrics reconstructed to justify a stage change already made are the clearest sign that the progression is running backward.
- **Platform work deferred in favor of more work classes.** A fifth work class standing on a thin substrate performs like the first four, and for the same reason. Adding classes is visible progress; deepening the substrate underneath them is not, which is why the second is the one that gets postponed.
- **Instrumentation that nobody reads.** The generated-to-analyzed ratio applies to an organization's own telemetry as much as to its agents' output.
- **Consolidation permanently deprioritized.** Duplication trends are slow, visible, and easy to defer for four consecutive quarters.

33.5 What This Rests On

The credential, provenance, and gateway items derive from documented failure modes in Part VI and from the control requirements in Sections 17 and 18. The mutation-testing and held-out-signal items rest on measured findings.[113,114,170] The two-quarter evidence requirement, the month boundaries, and the ordering within each period are this framework's construction. No organization has published a staged qualification timeline with outcome data against it, and the temporal-versus-logical control substitution in Section 33.2 is named as an open problem in Section 23 precisely because it is unmodeled.

PART VI

Case Studies

Eight cases, reconstructed entirely from published primary sources. Nothing here is a composite, an anonymized client engagement, or a scenario written to illustrate a point. Where a figure is derived rather than reported, the text says so. Where the published account is first-party and unaudited, the text says that too.

Each case is read through the same five questions. What was the context, including the denominators that make a percentage mean anything. What was actually built. What decided correctness—the oracle, in the sense Section 24 gives the word. What was measured, and what was conspicuously not. What transfers to an organization that is not the one in the case.

Two of the eight are failures. They are here because the successes share a property that only the failures make legible: in every case that worked, the thing that made it work was a machine-checkable decision procedure sitting outside the agent's reach, and in both cases that failed, that procedure was either absent or was the thing the attacker used.

§	CASE	ORGANIZATION	WORK CLASS	OUTCOME
34	Backlog work at repository scale	Microsoft, `dotnet/runtime`	Issue resolution	878 agent pull requests, 67.9% merged
35	Flaky-test repair as a deployed work class	Uber	Test repair	1,115 tests attempted, 17.7% end-to-end
36	Two migrations, two oracles	Google and Uber	Large-scale migration	Paired comparison
37	Assured test generation	Meta	Test generation	73% acceptance, 34-point inter-team spread
38	Fleet maintenance at scale	Spotify	Fleet-wide change	2.5M+ automated pull requests
39	Review at volume	Uber	Code review	90%+ of ~65,000 weekly changes
40	Agent tooling as attack surface	Nx ecosystem	Supply chain	Compromise
41	An agent outside its authorized scope	Replit and a customer	Production data	Destruction

34 Backlog Work at Repository Scale

The case. Ten months of a cloud coding agent operating inside one of the largest and oldest open-source repositories Microsoft maintains, reported with full denominators and a same-repository human control.

Source quality. First-party, vendor-adjacent: Microsoft reporting on the behavior of a Microsoft product in a Microsoft-governed repository.[105] It is nevertheless the most completely denominated public account of agent-authored pull requests that exists, and the author volunteers the selection-bias problem rather than being caught by it. No substantive independent critique of the report has appeared; four Hacker News submissions between March 23 and April 2, 2026 drew no comments at all. Treat it as unrebutted rather than as corroborated.

34.1 Context

`dotnet/runtime` is roughly nine years old as a repository and considerably older as a codebase, carrying lineage from `coreclr` and `corefx` before them. It spans managed and native code, targets multiple operating systems and processor architectures, and is maintained by a large permanent engineering staff alongside a substantial external contributor community. It is, in other words, the opposite of the environment in which agent benchmarks are run.

The reporting window runs from May 19, 2025 to March 22, 2026. Across that period the repository received 6,181 pull requests from all sources. The coding agent accounted for 878 of them, or 22.2 percent of everything Microsoft-originated.[105]

34.2 The Mechanism

An engineer assigns a GitHub issue to the agent. The agent provisions an ephemeral Linux environment, builds the repository, attempts the change, runs tests, and opens a pull request. Review then proceeds through the repository's ordinary process, with one significant difference: humans may push commits directly onto the agent's branch rather than only commenting on it.

That difference turns out to matter more than anything else in the report.

34.3 The Oracle

There is no purpose-built oracle in this case, and its absence is instructive. Correctness is decided by the repository's pre-existing continuous integration—a large authored test suite the agent did not write and cannot usefully modify without a reviewer noticing—followed by human code review.

This is the weakest oracle configuration that still works, and it works for a specific reason: the suite predates the agent by years, it is enormous, and the review culture around it is unusually strong. An organization with a thin suite and a permissive review norm cannot reproduce this result by adopting the same tool. The tool is not what is doing the work.

34.4 What Was Measured

AUTHOR CLASS	PULL REQUESTS	MERGED	RATE
Coding agent	878	535	67.9%
Microsoft engineers	3,082	2,556	87.1%
Community contributors	1,411	1,029	79.7%
Repository, all sources	6,181	4,786	82.7%

Across seven repositories the same agent produced 2,963 pull requests with 1,885 merged, a rate of 68.6 percent.[105]

Success varied sharply by task type: removal and cleanup 84.7 percent, testing 75.6 percent, refactoring 69.7 percent, bug fixes 69.4 percent, documentation 68.1 percent, upgrades 67.4 percent, features 64.5 percent, and performance work 54.5 percent. The ordering is the finding. Deleting code and writing tests beat writing features by roughly twenty points, and performance work—which requires reasoning about a machine the agent cannot measure—sits at the bottom of the legitimate categories.

Three further measurements deserve to be carried into any planning conversation.

Human commits into the agent's branch. Of 878 agent pull requests, 396 received direct human commits and 280 of those merged, a rate of 86.2 percent. The 482 that received none merged at 55.1 percent. The report's own phrasing is that success "jumps to 86%. Without that intervention, it's 55%."[105] Collaboration outperformed supervision by thirty-one points.

Review cost. A merged agent pull request drew 16.5 comments on average against 12.4 for a merged human one, with medians of 10 and 7. Heavy iteration—more than twenty comments—occurred on 18.9 percent of agent pull requests against 11.8 percent of human ones. Review was also concentrated: the top ten reviewers wrote 61 percent of all comments on agent pull requests, and the top two wrote 36 percent.

Reverts. Three of 535 merged agent pull requests were reverted, or 0.6 percent, against 33 of 4,251 merged non-agent pull requests, or 0.8 percent. This is the only direct quality signal in the report, and it does not show degradation.

The preparation effect is reported two ways, and they should not be merged. Measured by calendar, the success rate ran from 41.7 percent in the first month to approximately 71 percent across the most recent quarter. Measured by cohort, pull requests created before the first environment and instruction changes succeeded at 38.1 percent and those created after at 69 percent.[105] Either framing supports the same conclusion—that the change came from preparation rather than from model upgrades—but they are different comparisons and a document that quotes 41.7 to 69 percent is quoting neither.

What the preparation consisted of is enumerable, and it is mostly unglamorous. Firewall rules permitting access to the package feeds the build requires. A repository instruction file describing structure and build process. Guidance to use targeted rather than full builds. Documentation of the fact that the agent runs on Linux only, which is a hard constraint in a codebase with substantial Windows-specific implementation. Testing conventions, including the prohibition on asserting exact exception messages in a localized product. Architectural boundaries. Later, separate instruction files per agent, a rebuilt setup step that cut environment provisioning from over twenty minutes to a few, and a set of eight repository-specific skills covering benchmarking, regression testing, continuous integration failure analysis, and triage.

34.5 What Failed

Native and platform-specific code was persistently weak, for the structural reason that the agent could not compile or execute Windows-only paths from a Linux environment. Performance work produced unvalidated improvement claims until the agent was given a benchmarking harness on dedicated hardware. External contributions were out of reach entirely, because the source branch lives in a fork. Vague issues produced vague changes, and the agent did not extend beyond the scope it was handed.

Autonomy was more limited than the merge rate suggests: 62 percent of merged pull requests began with exactly two commits, meaning the agent made one attempt and then waited.

The report's most quoted passage is about the reviewer, not the agent:

> *"The bottleneck has moved. AI changes the economics of code production. One person with good judgment and a phone can generate PRs faster than a team can review them. This creates asymmetric pressure: the person triggering CCA work feels productive ('nine PRs!!'), while reviewers feel overwhelmed ('nine PRs??')."*
>
> — Stephen Toub, Microsoft, March 23, 2026

Two things are absent from the report and the author says so. Compute cost and continuous-integration resource consumption, including failed runs, are not analyzed. Downstream quality outcomes beyond the revert rate are not quantified.

And the earlier history is worse than the ten-month aggregate implies. In May 2025, within days of the agent being enabled, a set of its pull requests in this repository became a widely circulated example of the technology failing in public, including one implementing globalization comparison behavior that reviewers rejected for not accounting for locale-specific collation and that never merged.[243] That episode sits inside the 41.7 percent first-month cohort. It is the same program.

34.6 What Transfers

The finding with the widest application is the human-commit result, because it inverts the default operating assumption. Most organizations treat agent output as something to be judged: approve, request changes, or close. This repository's data says the higher-yield posture is to treat it as a draft to be finished, and that the difference is worth thirty-one points of merge rate.

The second transferable finding is that the improvement came from environment and documentation work that any organization can do and most have not. Package-feed access, a written description of how the repository is built, and an honest statement of what the agent's execution environment cannot reach are not advanced practice. They are the difference between 38 percent and 69 percent.

What does not transfer is the denominator. This repository has a mature suite, a strong review culture, and named reviewers with the standing to reject. Strip any one of those and the same tool produces the same volume with none of the assurance.

34.7 What This Rests On

Every figure in this chapter comes from a single first-party report, and its author's own caveat should be carried alongside them: agent pull requests "are not randomly sampled; they reflect deliberate choices about which tasks to assign to an AI agent," and comparisons against human pull requests are "between fundamentally different populations."[105] The 67.9 percent is not a capability measurement. It is the merge rate of the work someone chose to delegate.

35 Flaky-Test Repair as a Deployed Work Class

The case. A fully autonomous system repairing non-deterministic tests in a hundred-million-line monorepo, reported with a complete funnel and an unusually candid rejection analysis.

Source quality. Peer-reviewed and industrially deployed: a distinguished-paper award at ASE 2025, co-authored by university and Uber researchers, describing a six-month production deployment.[161] This is the strongest evidentiary position of any case in Part VI. It is also the only one whose headline number is materially misleading if read without the funnel beneath it.

35.1 Context

The codebase is Uber's Go monorepo: approximately 100 million lines across more than 100 projects, worked by over 6,000 engineers on more than 100 teams. The baseline problem is well characterized and predates agents by a decade. At Google, roughly 1.5 percent of all test runs report a flaky result, almost 16 percent of tests exhibit some flakiness, and about 84 percent of observed pass-to-fail transitions involve a flaky test.[162] A flaky suite does not merely waste time; it destroys the signal that any delegation strategy depends on.

35.2 The Mechanism

The system is wired into the existing ticketing system and delivers fixes to developers daily without human initiation. Its contribution is not the repair prompt. It is context selection.

Compile-time instrumentation builds a dynamic call graph from actual test execution, which contains roughly 40 percent of the nodes a static call graph would. A model-guided breadth-first traversal then expands only the paths judged relevant, on the observation that the information needed to explain a flake usually sits in leaf nodes deep in the graph rather than near the test. The authors name the problem they are solving directly: prior approaches fail in industrial settings by supplying "either too little context (missing critical production code) or too much context (overwhelming the LLM with irrelevant information)."[161]

Reproduction is brute force rather than clever. The test is executed a thousand times with Go's race detector enabled to perturb scheduling; if it never fails, the entire test target is executed a thousand times and the results filtered. The repair budget is three context-collection attempts, two reasoning paths, and three candidate fixes per path—eighteen attempts per test, under a two-hour cap.

35.3 The Oracle

Build validation, then test validation: the repaired test is rerun the same number of times used for reproduction, and every run must pass. Any error or timeout reverts the candidate and advances to the next.[161]

This is a statistical oracle, and it certifies exactly one property: the flakiness stopped. It cannot certify that the test still tests what it tested. That gap is not hypothetical, and the rejection analysis in Section 35.5 is what happens when it opens.

The system prompt constrains the agent to modify test code only, never production code. That constraint is the reason the oracle holds at all—an agent permitted to edit the code under test could make any test deterministic.

35.4 What Was Measured

STAGE	COUNT	RATE
Flaky tests attempted	1,115	—
Reproduced	798	71.6%
Fixes produced	380	47.6% of reproduced
Fixes accepted and landed	197	51.8% of produced
End to end	**197 of 1,115**	**≈17.7%**

The end-to-end figure is arithmetic, not a reported result; the paper states the three conditional rates and does not multiply them.[161] It should nevertheless be the planning number. An organization that budgets against 51.8 percent will under-provision by roughly a factor of three.

There is a second gap worth naming. In a controlled evaluation on 295 reproducible tests at a single commit, the system repaired 65.76 percent, against 45.42 percent for the strongest prior tool. In six months of live deployment the same metric was 47.6 percent. That is an eighteen-point benchmark-to-production drop on the same measure, before any human acceptance filter is applied. The paper reports both figures and does not draw the comparison. It is available and it is the single most useful number in the paper for anyone sizing a pilot from published results.

Root causes among accepted fixes: scheduling randomness 37 percent, unordered collection iteration 33 percent, timestamp discrepancy 12 percent, state pollution 8 percent, time dependence 7 percent, other 3 percent.

Mean repair time was 1,978.73 seconds per test—roughly thirty-three minutes—which made this the slowest of the three systems compared, notwithstanding its higher yield. Token counts and monetary cost are not reported.

35.5 What Failed

Of 380 produced fixes, 183 were rejected. The stated reasons, in the authors' order of importance:

1 The fix did not use the organization's custom test-helper APIs, which the model had no knowledge of.

2	The fix was at the wrong difficulty level—"the easier ones are more likely to be produced and sent to developers, but the more complex fixes are more proper and what developers want." The worked example is a mock relaxed to `.MaxTimes(1)`, which passed validation while weakening what the test asserted.
3	The fix altered test semantics: "removing some assertions, adding irrelevant test assertions."
4	The fix was too long or too complex to read.
5	A developer had already fixed the test by hand and preferred their own version.

Reasons two and three are the ones that matter for control design. Both describe fixes that **passed the automated oracle and were still wrong**. Only human review caught them. A deployment of this work class that treats oracle passage as sufficient will land assertion removal at scale, and the resulting suite will be green and worthless.

A developer survey returned nineteen responses. All nineteen found the root-cause explanations useful; mean fix-quality rating was 4.42 of 5. The most common time-saving estimate, at 42.1 percent of respondents, was *less than one day per test*—which is a useful corrective to the way such systems are usually sold.

35.6 What Transfers

The reproduction requirement transfers first and hardest: 28.4 percent of reported flaky tests never reproduced, and nothing downstream can help them. Any organization planning this work class should measure its own reproduction rate before anything else, because that number caps everything.

The guardrails transfer next, and they follow directly from the rejection analysis. Assertion-count comparison and mutation score on the repaired test, both computed automatically and both gating, are the minimum defense against a fix that makes a test green by making it empty. A green build is not evidence here; it is the thing the failure mode produces.

Last, **the custom-API rejection reason is a context-substrate problem, not a model problem.** The organization's own testing conventions were invisible to the system, and roughly a fifth of its rejections followed. Section 28 is the remedy.

35.7 What This Rests On

The funnel, the oracle description, the rejection reasons, and the root-cause distribution are all reported in a peer-reviewed paper. The 17.7 percent end-to-end figure and the eighteen-point benchmark-to-production gap are derived by this framework from figures the paper reports separately. The per-reason counts behind the rejection analysis are not published, so the claim that assertion removal is common rests on the authors' qualitative ranking rather than on a count.

36 Two Migrations, Two Oracles

The case. The same category of engineering work—mechanical transformation applied across an enormous codebase—solved two ways at two organizations, with an order-of-magnitude difference in throughput that is explained almost entirely by what sat in the validation loop.

Source quality. Mixed and unusually good. The Google side is two peer-reviewed papers with declared threats to validity.[107,166] The Uber side is two first-party engineering posts, unaudited, and one of them contains a candid statement against interest.[168,234] The comparison itself is this framework's construction; neither organization presents its work as a contrast with the other.

36.1 Why These Two Are Paired

Large-scale migration is the work class where agentic engineering is most often assumed to be obviously superior, because the work is repetitive, well specified, and enormous. It is also the work class with the best published evidence on both sides of the method question, which makes it the only place in the current literature where a controlled-ish comparison is possible.

The pairing is not "AI versus no AI." Both organizations use language models heavily and both were doing so at the time. The pairing is about where the model sits in the pipeline, and what decides whether a transformed artifact is correct.

36.2 Google: The Model Generates, Humans Decide

Google's published migration work covers a twelve-month effort by three developers to migrate 39 distinct identifiers from 32-bit to 64-bit across an advertising codebase exceeding 500 million lines, plus three other migration families reported separately.[107,166]

The architecture divides labor explicitly. Finding the places to change is deterministic: the Kythe code index resolves direct references, then indirect references to a maximum distance of five. Categorizing them is deterministic, by regular-expression classifiers into buckets for definitely-changed, definitely-irrelevant, needs-investigation, and manual-review. Only the edit itself is model-generated, three candidate modifications per file with one chosen at random from the successful attempts.

The authors state the division plainly:

> *"The LLM role is focused on the edit generation. The parts where we need to identify locations at which to make changes, and where we need to validate that the right thing took place, are handled mostly using deterministic AST techniques with a few cases supported by the LLM as well."*
>
> — Nikolov et al., Google, January 2025

The oracle is a stack of six gates, and the last one is a person. A candidate change is invalid if the model returned no completion, if it changed only whitespace, if the result fails to parse or produces an identical syntax tree, if a second model call judges the change unnecessary, if the build breaks, or if regression tests fail. Everything surviving that is then reviewed visually by a developer, repaired by hand where needed, and sent to the owning team for acceptance.[166]

The outcome across the identifier migration: 595 changes containing 93,574 edits, of which 69.46 percent of edit distance was model-authored. At the level of whole changes, 35.97 percent landed with no human edit at all, 38.48 percent were model-generated and then human-edited, and 25.55 percent were written by hand. Review surface was 306 reviewers across 149 teams, 37 offices, and 12 time zones. The prior manual attempt at a single identifier had taken around two years.[166]

Two definitional points govern how these numbers may be used. First, success in this program is throughput, not quality: "for each migration described we have defined success as AI saving at least 50% of the time for the end-to-end work," including finding sites, reviewing, and rolling out.[107] Second, the 50 percent is self-reported. The paper's own threat-to-validity section states that it "is based on developer perception and estimations, and not on precise time tracking data."[166]

And the constraint that binds is named without hedging:

> *"The bottleneck in the process was the speed at which engineers could review the changes. We purposefully limited the number of changes we generate every weak [sic] to avoid overwhelming reviewers."*
>
> — Nikolov et al., Google, January 2025

An organization with a review-shaped oracle throttles generation to protect the oracle. That is not a failure of the tooling. It is the only rational response to the architecture.

36.3 Uber: The Machine Decides, at Population Scale

Uber's Spark upgrade covers a different unit of work. Roughly 2 million Spark applications launch daily through more than 20,000 scheduled workflows, drawing on more than 2,100 applications of Spark source code.[168] The three numbers are stated separately in the source and never reconciled; the 2 million is a daily execution count, not two million distinct programs, and the distinction matters when comparing against Google's change counts.

Code transformation used Polyglot Piranha, an open-source abstract-syntax-tree rewriter Uber originally built for stale feature-flag cleanup, extended with structural rules for the Spark 2 to Spark 3 transition. No language model appears anywhere in the account. That is worth stating precisely: the post does not argue against using one, it simply never raises the possibility.

The oracle is the interesting part, and it is nothing like Google's. Uber built a framework called Iron Dome that validates a transformed job by *running it against production data and comparing what it wrote*. Spark's catalog interface and Hadoop's file output committer are intercepted so that a production path `/db/tbl` is rewritten at runtime to `/stgdb/tbl`. Guardrails at the filesystem interface prevent accidental production writes. The

interceptors publish telemetry naming every table and path the job touched, and that telemetry is used to validate output against the production counterpart. Orchestration then shadows production runs, validates the data, and marks jobs for migration automatically.[168]

No human reads a diff. The oracle is output equivalence on real data, checked by machine, for every artifact.

Reported outcomes: an "85% job migration" heading, with the sentence beneath it stating that the majority of the 20,000 jobs transitioned within six months; more than 60 percent of jobs improving by more than 10 percent in performance; and eventual completion at 100 percent of Spark applications upgraded to 3.3, with a 50 percent reduction in runtime and resource usage overall.[168] The fraction of the 2,100 source applications that Piranha transformed without manual work is not published, and should not be inferred from the 85 percent, which is a completion figure rather than an automation rate.

36.4 The Same Migration, Both Ways

The Spark comparison is imperfect because the units differ. A second Uber account closes that gap, because it covers the same migration family Google reported—JUnit—and states outright that language models were tried and set aside.

Uber migrated more than 75,000 test classes from JUnit 4 to JUnit 5, modifying over 1.25 million lines within four months, against a denominator of more than 600,000 JUnit 4 tests across 15 million lines. The transformation ran on OpenRewrite recipes operating over a lossless semantic tree, integrated with the build system through custom Bazel aspects, producing over 5,000 diffs.[234]

The key learning is stated in three sentences:

> *"We attempted to use generative AI to migrate multiple test class files at once, but this approach was unsuccessful. AI usage was limited to debugging test and build failures. Given our codebase's scale and custom testing patterns, a deterministic approach proved more practical than an AI-driven migration."*
>
> — Uber Engineering, April 7, 2026

Note that this is precisely the inverse of Google's split. Google puts the model on edit generation and the deterministic machinery on location and validation. Uber puts the deterministic machinery on edit generation and the model on failure triage.

The failure-handling policy follows from the scale: "For test failures, we reverted affected files back to JUnit 4. Given the scale spanning thousands of targets, diagnosing individual test failures was impractical." Recurring build failures were addressed by writing additional recipes—fixing the rule rather than the instance.[234]

For comparison, Google's JUnit 3 to JUnit 4 migration covered 5,359 files and more than 149,000 lines in three months, with approximately 87 percent of model-generated code committed unchanged.[107]

36.5 What the Pair Shows

DIMENSION	GOOGLE, LLM-ASSISTED	UBER, DETERMINISTIC
Edit generation	Fine-tuned model, 3 candidates per file	AST rewriter, rule-expressed
Location finding	Deterministic index (Kythe)	Deterministic, build-integrated
Validation	Build, tests, model-as-judge, then human review	Shadow execution and output comparison (Spark); build and test with revert-on-failure (JUnit)
Binding constraint	Reviewer bandwidth, explicitly throttled	Rule-authoring effort
Unit of throughput	595 changes, 93,574 edits, 12 months	20,000 workflows in 6 months; 75,000 test classes in 4 months
Quality assurance	Existing human review process; long-term effect stated as unknown	Machine-checked output equivalence, or revert
Failure handling	Human repairs the change	System reverts the file, engineer fixes the rule

The generative capability is not what separates these programs. What separates them is that one has an oracle a machine can run on every artifact and the other has an oracle made of people.

Where a cheap mechanical decision procedure exists—output equivalence, differential comparison against a reference, a rule that is correct by construction—determinism scales to population size and the model is best used on the residue. Where no such procedure exists, the model can still generate at volume, but throughput is capped by review, and the responsible organizations throttle generation to match. Google did exactly that and said so.

This is the same finding as Section 6.1, arrived at from the opposite direction. Delegate by verifiability, not by difficulty.

36.6 Two Cautions

The unit gap is partly an artifact. Google counts changelists and edits authored; Uber counts jobs executed and validated. A Spark job cleared by an output diff after a configuration-injection rule is not the same quantum of work as a semantic type migration through a call graph five hops deep. The honest contrast is not 595 against 20,000. It is what sits in the validation loop.

Uber is not the deterministic foil this pairing might suggest. By August 2026 the same organization attributes more than 70 percent of its pull requests to local or cloud agents, runs more than 3,600 agent skills executing over 30,000 times a day, and reports weekly agent requests up 9.4 times in six months.[242] Both accounts above predate or coincide with that build-out, and the JUnit post keeps the model in the loop for failure triage. The defensible reading is method selection by problem class, not a verdict on agents.

36.7 What Transfers

Before delegating a migration to agents, answer one question: *does a mechanical check exist that can decide correctness for every artifact without a person reading it?* Shadow execution with output comparison, differential testing against the prior version, and a transformation that is correct by construction all qualify. Build-plus-tests qualifies only where the suite is strong enough that passing means something.

If such a check exists, build it first and let it govern the rollout, whatever generates the edit. Uber's Iron Dome is the reusable artifact from that migration, not the Piranha rules; the post says so, noting the framework "paved the way for future Spark upgrades."[168]

If no such check exists, size the program against reviewer capacity from the first week, and throttle generation deliberately rather than discovering the ceiling by overwhelming a team. Google's throttling is the practice to copy.

36.8 What This Rests On

The Google figures are peer-reviewed and carry the authors' own generalizability warnings: a single organization's infrastructure, a single migration type, a model fine-tuned on internal code, and effort measured by edit distance as a proxy. The 50 percent time saving is developer estimation, not instrumentation. The Uber figures are first-party and unaudited, and the Spark post's headline percentage is a section heading rather than a measured automation rate. Neither organization publishes a defect-escape rate, so no claim about relative correctness is available from either side—only about relative throughput and about what decided correctness along the way.

37 Assured Test Generation

The case. Two successive systems at Meta that generate tests only when a machine can prove the test is an improvement, and a thirty-four-point acceptance gap between two teams using the identical tool.

Source quality. Industry-track experience reports at FSE Companion, not full research-track papers with independent artifact evaluation, and first-party throughout.[169,170] They are unusually forthcoming about their own limits, which is why they are here. No peer-reviewed replication or rebuttal exists; the only substantive independent engagement is an open-source reimplementation by a vendor with a competing product.[241]

37.1 Context

The methodological frame the authors call Assured Offline LLM-Based Software Engineering asks two questions of any model-generated change: does it avoid regressing the properties of the original, and does it improve on the original in a verifiable, measurable way.[239] Test generation is where those questions have crisp mechanical answers, which is why it was the first deployment.

The first system augments an existing test class rather than writing a new one. That single design choice supplies the non-regression guarantee for free: "TestGen-LLM simply augments an existing test class with additional test cases, retaining all existing test cases and thereby guaranteeing there will be no regression, by construction."[169]

37.2 The Mechanism and the Oracle

Here the filter chain *is* the oracle, and it is the only case in Part VI where the generation step is treated as interchangeable and the assurance step is treated as the product.

Three sequential filters, each of which discards everything that does not pass. The candidate must build within the existing infrastructure. It must pass on first execution and on every one of five subsequent executions, or it is deemed flaky and discarded. It must increase coverage over the original class, or it is discarded.[169]

The second system replaces the coverage filter with something stronger. A model generates a small number of concern-specific faults—mutants—in the class under test, guided by a plain-text description of the concern and a differential example of a real past fault. A second model acts as an equivalence detector, discarding mutants semantically identical to the original. A third generates a test that must **fail on the mutant and pass on the correct code**.[170] That is a genuine held-out signal in the sense of Section 24: the criterion is fault detection, and it is not something the generating agent can satisfy by writing an empty test.

The authors of the follow-up work are careful about which of their six assurances are real:

> *"The first four of these six assurances are assurances in the strict meaning of the word: they are verifiable (and, therefore, falsifiable) guarantees that leave no room for doubt."*
>
> — Harman, O'Hearn, and Sengupta, Meta, 2025[240]

Buildability, non-flakiness, fault detection, and coverage reporting are machine-decidable. Relevance to the concern and conformity to house style are not. The gap between those two groups is where the interesting failure in Section 37.4 lives.

37.3 What Was Measured

The first system's filter funnel, measured across 86 Kotlin components with existing human-written test classes: 75 percent of test classes yielded at least one new case that built correctly, 57 percent at least one that built and passed reliably, and 25 percent at least one that built, passed, and increased line coverage.[169] These are cumulative class-level rates, not per-test-case pass rates, and the paper's abstract states them more loosely than its body does.

At deployment across three test-a-thons between October 29 and December 29, 2023, the system was applied to 1,979 test classes and successfully improved 196 of them—roughly 10 percent. Of the improvements it did produce, 73 percent were accepted by developers.[169] A note for anyone citing this: the paper's abstract says 11.5 percent where its body says approximately 10 percent, and 196 of 1,979 is 9.9 percent. The counts are the reliable part.

At the level of individual model attempts, the ratios are starker. Of 8,996 trials on one platform, 490 produced a surviving test; of 23,535 on another, 831 did. Roughly four to five percent of generation attempts survive the filters. **The system's value is almost entirely in what it throws away.**

The second system ran from October 28 to December 31, 2024 across 10,795 Android Kotlin classes on seven platforms. It generated 31,677 mutants, of which 9,095 built and passed. Of those, 2,246 were syntactically identical to the original, 1,016 were judged equivalent, 1,173 produced no judgment, and 4,660 were usable. From those it produced 571 hardening test cases—294 of which added line coverage and 277 of which did not.[170]

That last split is the most consequential number in either paper:

> *"Approximately half of all tests generated, although finding faults missed by all existing tests, do not add line coverage. Had we used coverage as our sole test adequacy criterion, the platforms would thus have continued to be vulnerable to regressions denoted by up to approximately half of the faults."*
>
> — Foster et al., Meta, 2025

The filter you choose determines what you catch. A coverage gate would have discarded half the useful output of a system explicitly designed to find faults.

37.4 What Failed, and the Thirty-Four-Point Spread

Overall acceptance across 191 reviewed tests was 73 percent. Disaggregated by team, on the same tool, the same model, and the same generation pipeline:

REVIEW EVENT	REVIEWED	ACCEPTED	RATE
WhatsApp	91	50	56%
Messenger, phase 1	50	47	94%
Messenger, phase 2	50	43	86%
Messenger combined	100	90	90%
Total	**191**	**140**	**73%**

Thirty-four percentage points separate the two products. The authors checked the obvious explanation and it does not hold: WhatsApp engineers rated the tests *more* relevant to the privacy concern than Messenger engineers did, at 39 percent against 33 percent, and still accepted far fewer. Their explanation is the honest one:

> *"We believe that the different acceptance rates may simply denote different cultures between different teams; deciding to land a test into production is an inherently subjective judgment and may be influenced by team culture."*
>
> — Foster et al., Meta, 2025

This is the strongest published evidence in Part VI that **acceptance rate measures the receiving organization, not the tool.** Any pilot that reports a single acceptance number across multiple teams has averaged away the variable that actually determines the result.

Two secondary findings sit alongside it. Only about 36 percent of accepted tests were judged relevant to the privacy concern that motivated them, and engineers accepted them anyway because they were useful for other reasons. And the team's own pre-screening turned out to be wrong: ten tests were excluded in the first phase on the assumption that null-pointer-exception tests would be rejected as irrelevant, and when pre-screening was dropped in the second phase, acceptance stayed high.

The independent critique available is narrow but fair. The evaluation is Kotlin only; whitespace-sensitive languages stress the build filter differently. Yield of roughly 10 percent is the real-world ceiling, not the 75 percent that gets quoted. And the system is not autonomous—it requires human review between iterations.[241]

Neither paper publishes cost, token counts, or compute.

37.5 What Transfers

Adopt the augment-only constraint wherever it applies. Generating additions to an existing artifact rather than replacing it converts non-regression from something you have to test into something that is true by construction. It is the cheapest assurance in this document.

Replace coverage with fault detection as the gate on agent-written tests. The 294-to-277 split is the evidence, and Section 24.4 gives the procedure.

Expect and instrument the team spread. Before scaling a test-generation program, run it in two teams and compare acceptance. If the gap is wide, the intervention is review-norm alignment, not prompt engineering.

And **treat generation yield as a cost line rather than a headline.** Four to five percent trial survival is a compute budget, and neither paper reports what it cost.

37.6 What This Rests On

Both papers are first-party industry-track reports without independent artifact evaluation. The funnel figures, the team acceptance split, and the coverage-versus-fault-detection result are directly reported. The internal inconsistency in the first paper's improvement rate is unresolved in the source. The claim that acceptance measures the organization rather than the tool is this framework's reading of the authors' own team-culture explanation, and it rests on a single two-team comparison with roughly twelve reviewers in total.

38 Fleet Maintenance at Scale

The case. Seven years of deterministic fleet-wide change infrastructure at Spotify, and what happened when an agent was layered on top of it.

Source quality. First-party engineering blog posts throughout, spanning 2023 to 2026, with self-reported adoption and no independent audit.[100,101,235,236,237,238] Quality outcomes and failure rates are largely absent, which is a real limitation on what can be concluded.

38.1 Context

Spotify runs a polyrepo layout: thousands of GitHub repositories holding small, independently deployable components owned by individual squads. Roughly 60 million lines sit in production components. The condition that forced the investment is stated plainly—the production codebase was growing seven times faster than the engineering headcount.[101]

38.2 The Mechanism

The unit of change is not a patch. It is a container image.

A “shift” is a Kubernetes custom resource stored in GitHub that names a Docker image containing the transformation logic, the target repository set, and the pull request metadata. The platform clones each repository, executes the image against the checked-out code, records the file changes, creates a commit, and opens a pull request. Targeting is done by GitHub search or by querying ingested source code in a data warehouse. Because the transformation is a container, it can be anything—stream editors, purpose-built code, `OpenRewrite`, `Scalafix`—and the platform does not care which.[235]

Merge authority is inverted. An automerger service merges pull requests that pass all tests, which moves the decision from the repository-owning team to the change author. Two guards constrain it: automerge runs only during the owning team’s working hours and never on weekends or holidays, and risky changes roll out in cohorts, with each cohort gated on the previous one succeeding. A service consuming deployment and pipeline-execution events correlates failures with recently automerged changes and feeds that signal back into the gating.[235]

None of this used a language model. The 2023 account describes ordinary tooling and custom code, and mentions no AI at all.

38.3 The Oracle

The repository's own tests, plus production telemetry as a second-order check.

That is thinner than the Uber Spark oracle in Section 36 and much thinner than Meta's in Section 37, and the compensating mechanism is temporal rather than logical: cohort gating means a defective change is caught after it has damaged a small number of repositories rather than before it has damaged any. The working-hours constraint exists for the same reason—so that a human is present when the cohort fails.

This is a legitimate design. It is not the same thing as verification, and an organization adopting it should be clear about which one it is buying.

38.4 What Was Measured

In 2022, the fleet platform created more than 270,000 pull requests, of which 77 percent were automerged and 11 percent merged by a human. The 241,000 merged changes represented 4.2 million lines changed.[235] Time to reach 70 percent fleet adoption of a backend framework release fell from about 200 days to under seven. During Log4j, 80 percent of backend services were patched within nine hours.

By June 2026 the cumulative figure was "more than 2.5 million automated maintenance PRs," described as "the vast majority auto-merged with no human in the loop."[101] No start date is given for that total, so it should be cited as cumulative-to-2026 rather than attributed to any single period, and the boundary between deterministic and agent-generated changes within it is not published.

The agent layer arrived later, with an explicitly stated purpose: to lower the barrier to entry so that fleet management could handle changes too complex to express as a rule.[100] By November 2025 it had produced more than 1,500 pull requests merged into production, with reported time savings of 60 to 90 percent against writing the same code by hand and hundreds of developers interacting with it. No merge rate—merged over opened—is published for the agent at any point in the four-part series, which is a conspicuous omission given how carefully the deterministic platform's rate was reported.

One worked deployment is reported in full: roughly 1,800 downstream data pipelines affected by deprecated datasets, 240 automated migration pull requests deployed over six months across three pipeline frameworks, against an estimate of about ten engineering weeks to do the same work manually.[238]

38.5 What Failed

The engineering team's own account of what did not work is the most useful part of the series.

Their homegrown agent loop, capped at ten turns per session with three session retries, exhausted its turns on multi-file cascading changes. Open-source agent frameworks "struggled producing reliable, mergeable PRs at scale." Requiring users to name exact files created friction. Tool proliferation introduced unpredictability, and limiting the agent's tool access improved reliability—the agent was eventually given a verification tool, a restricted git tool, and a shell tool on a strict allowlist, with code search and documentation tools deliberately withheld.[236]

A model judge vetoes roughly 25 percent of agent sessions, and about half of vetoed sessions result in self-correction. The stated reason is worth quoting because it names a failure mode most organizations do not instrument: “some agents were a bit too ‘ambitious’, trying to solve problems that weren’t strictly in their prompt.”[237]

Their ranking of failure modes matches this framework’s: an agent that produces no pull request is an annoyance; one that produces a failing pull request is frustrating; one that produces a pull request that passes continuous integration and is functionally wrong is the serious case, because it erodes trust.[237]

Verification tooling supports Linux on x86 only, which excludes iOS and ARM work entirely.

And the aggregate effect on the organization is stated without spin: a 76 percent increase in pull request frequency means “76% more PRs to review.”[101] Generation scaled. The review function absorbed all of it.

38.6 What Transfers

The structural finding is the sequencing. Seven years of deterministic infrastructure—targeting, cohorting, automerge policy, failure correlation—is what made an agent deployable here. The agent inherited a rollout and verification system it did not have to build. An organization without that substrate that adopts a background agent is not reproducing this case; it is reproducing the first year of it.

The tool-restriction finding transfers directly and is cheap to apply: fewer tools produced more predictable agents, and the tools withheld were the open-ended search ones.

The judge-veto rate is worth adopting as an instrument even where the judge itself is weak. Twenty-five percent of sessions attempting something outside their brief is a scope-discipline measurement, and almost nobody computes it.

The automerge design transfers only with its guards intact. Automerge without cohort gating, without working-hours constraint, and without failure correlation is not this system.

38.7 What This Rests On

Everything here is first-party and self-reported. The 2.5 million figure has no published time window and no published composition. No failure rate, rollback rate, or defect-escape rate is published for automerged fleet changes at any point in the series, which means the safety of the automerge design is asserted rather than demonstrated. The agent’s merge rate is unpublished. The 60 to 90 percent time savings is an estimate.

39 Review at Volume

The case. A code-review system covering more than 90 percent of an organization's weekly changes, whose published comparison against human reviewers is weaker than it first appears.

Source quality. A single first-party engineering post with no independent evaluation, no follow-up, and no conference talk.[108] It is included partly because the system is real and instructive, and partly because its headline comparison is a good illustration of how an honest-looking control can fail.

39.1 Context and Mechanism

Uber's review system analyzes over 90 percent of roughly 65,000 weekly changes across six monorepos, processing more than 10,000 commits a week excluding configuration files, at a median latency of four minutes within continuous integration.[108]

The architecture is four stages. Ingestion filters low-signal targets—configuration, generated code, experimental directories—and constructs prompts with surrounding code context. Generation runs three specialized assistants: one for bugs, exception handling, and logic flaws; one enforcing organization-specific conventions from a shared registry; one for application security. Post-processing applies three filters: a second model call assigns each comment a confidence score, a semantic-similarity filter merges overlapping suggestions, and a category classifier suppresses comment types with historically low developer value. Feedback collection closes the loop.

The generation model and the grading model are different providers. That is the closest thing here to independence between production and judgment, and it is a deliberate design choice rather than an accident of procurement.

The suppression philosophy is stated directly: "A high false-positive rate undermines engineers' perception of the tool's accuracy and usefulness—when they encounter many false-positive comments, they start to tune out and ignore them." Categories consistently rated poorly are suppressed outright: readability nits, minor logging tweaks, low-impact performance suggestions, and stylistic issues.[108]

39.2 The Comparison, and Why It Is Not a Control

Engineers mark 75 percent of posted comments useful. On automated evaluation, an average of 65 percent of posted comments are addressed in the same changeset. The post then states that internal audits show only 51 percent of human-written comments are "considered as bugs by the author and addressed in the same changeset," and concludes that the system's "performance significantly exceeds that of human reviewers."[108]

The two numbers are produced by different methods against different criteria. The 65 percent is measured automatically, by re-running the review system five times against the final commit and counting a comment as resolved when none of the re-runs reproduces a semantically similar comment. The 51 percent comes from human audit and carries a compound condition—the author had to consider the comment a bug *and* address it in the same changeset. The population, sample size, time window, and team scope of that audit are not published.

This matters beyond pedantry. The framework has repeatedly treated this figure as the most credible published claim that agent review adds signal, on the strength of its apparent same-population human baseline. That reading was too generous. What exists is two figures measured differently, and a first-party interpretation of the gap between them. The comparison is suggestive. It is not a control.

The system's own limits are stated candidly, which counts in its favor:

> *"uReview today only has access to the code, and not to other artifacts like past PRs, feature flag configurations, database schemas, technical documentation... it can't correctly assess overall correctness and review the system design. It's much better at catching bugs that are evident from analyzing the source code alone."*
>
> — Uber Engineering

39.3 What Is Missing

No recall figure, no false-negative rate, and no statement of what proportion of real defects the system misses. Uber reports building precision-recall dashboards internally and publishes neither number. No ratio of candidate comments generated to comments posted, which means the suppression rate—the thing the whole design turns on—is unquantified in public. The 1,500 hours saved weekly, and the 39 developer-years annually derived from it, come from an internal benchmark asserting that a second human reviewer would spend ten minutes per commit; it is a modeled figure, not a measurement.

39.4 What Transfers

Engineering for scarcity rather than coverage is the transferable design principle, and it is the same principle Section 6 draws from the wider review literature. A system that comments on everything it could comment on trains its readers to ignore it.

Splitting generation and grading across different model providers is a cheap approximation of independence, and it is available to any organization with two vendor relationships.

The measurement lesson is the one to internalize most. If an organization intends to claim its review agent beats human reviewers, the two figures must be produced the same way, on the same population, over the same window. Otherwise the honest claim is narrower: comments the system posts are addressed at a high rate, and the system covers nearly everything, which is worth having on its own terms.

39.5 What This Rests On

One first-party post, unaudited, with no independent replication. Every figure in Section 39.1 is directly reported. The critique in Section 39.2 rests on the post's own description of its two measurement methods, which differ on its own account. The absence of recall data is stated by omission and should be treated as unknown rather than as poor.

40 Agent Tooling as Attack Surface

The case. A compromised build-tool package that used the developer's own locally installed coding agents as reconnaissance instruments, and what the refusal data tells us about where the boundary actually held.

Source quality. Strong and multi-sourced: a first-party postmortem and security advisory from the affected vendor, plus independent analyses from four security firms with differing incentives and differing conclusions.[59,60,61,224,225,226,227] The scale figures conflict materially between sources and the conflict is reported below rather than averaged away.

40.1 What Happened

On August 21, 2025 a workflow was merged into the Nx repository that interpolated a pull request title directly into a shell command. Combined with a trigger that runs with the target branch's elevated permissions, a repository token that still carried GitHub's legacy read-write default, and a publish workflow that could be invoked programmatically, the result was a complete path from an attacker-authored pull request title to package publication rights.[224]

The vulnerability was disclosed publicly on 21 August and the workflow reverted on 22 August. The attacker had already used it. On 24 August they committed a modified publish workflow to a branch, triggered it through the API, exfiltrated the registry token, and deleted the workflow run from the audit log. On the evening of 26 August, eight versions of the primary package and several scoped packages were published to the registry over roughly two hours.[224,226] The registry removed them and revoked publish tokens shortly after 02:44 UTC on 27 August.

The package has roughly six million weekly installs. Exposure was not limited to direct installation: transitive dependencies, an editor extension that fetched the latest version to perform a version check, automated dependency-update scripts, and agent-initiated installs all reached the malicious `postinstall` script.[224]

40.2 The Payload, and the Part That Matters Here

The script fingerprinted the host, then read what it could find directly: the GitHub CLI token, registry credentials, SSH private keys, environment files, and cryptocurrency wallet artifacts. It merged everything into a single blob, triple base64-encoded it, authenticated to the GitHub API with the victim's own stolen token, created a **public** repository under the victim's account, and uploaded the blob there. It then appended a shutdown command to the user's shell profiles.[60]

Between the credential theft and the exfiltration, it did something new. It checked for three locally installed coding agents by name and, where present, invoked each with that product's own permission-bypass flag — `--dangerously-skip-permissions` for one, `--yolo` for another, `--trust-all-tools --no-interactive` for the third—passing a prompt instructing the agent to enumerate the filesystem and write a newline-separated inventory of interesting file paths to a temporary file.[225,226]

The prompt's construction is the instructive part. It opened by asserting that the agent was "an authorized penetration testing agent" operating "with explicit permission and within the rules of engagement," and it explicitly instructed the agent *not* to open, read, move, modify, or exfiltrate anything—only to list paths.[60] The malware did the reading itself. The agent was asked to do something that, taken in isolation, looks almost benign.

An assessment named this correctly at the time:

> *"To our knowledge, this is one of the first documented cases of malware coercing AI-assistant CLIs to assist in reconnaissance."*
>
> — Ashish Kurmi, StepSecurity, August 27, 2025[228]

40.3 What the Agents Actually Did

This is the most valuable data in the case and it is routinely omitted from summaries.

The agents did not reliably comply. Post-incident analysis found that almost a quarter of interactions with one agent were rejected outright; another was foiled in roughly a quarter of cases by default restrictions confining it to workspace directories; and across the population, the agent-assisted path "only exfiltrated data successfully in under a quarter of cases."[227]

The reason for the refusals is worth stating precisely, because it cuts against the attacker's own cleverness. The jailbreak framing triggered the guardrail: "The introduction of the phrase 'penetration testing', for example, was concretely reflected in LLM refusals to engage in such activity."[227] The social engineering that made the request look legitimate to a human reader is what made it look illegitimate to the model.

Additional friction had nothing to do with safety at all: account credit exhaustion, credential problems, one agent's permission-bypass flag still requiring an approval step, and tool-configuration failures.[227]

The yield was correspondingly poor. Under 100 unique valid secrets across roughly 20,000 exfiltrated files, mostly credentials for AI services and cloud platforms, with no observed successful cryptocurrency exfiltration despite the second prompt variant targeting wallets specifically.[227]

40.4 What Was Measured, and Where the Sources Disagree

MEASURE	FIGURE	SOURCE
Distinct stolen secrets, first phase	2,349 across 1,079 compromised systems	GitGuardian[61]
Credentials still valid at assessment	over 1,100	GitGuardian[61]
Compromised systems running macOS	85%	GitGuardian[61]

MEASURE	FIGURE	SOURCE
Victims with at least one agent installed	roughly 50%	Wiz[227]
Systems with an agent client installed	33%; 95 of 366 agent interactions succeeded	GitGuardian[61]
Public repositories created for exfiltration	over 1,400	GitGuardian[61]
Private repositories flipped public, second phase	5,500 → 6,700 (revised)	Wiz[227]
Private repositories flipped public, second phase	approximately 10,767, containing 82,901 secrets of which 11,168 valid	GitGuardian[61]

The two independent analyses disagree by roughly a factor of two on agent prevalence and by a similar margin on second-phase repository counts, and one of them revised its own figure upward mid-investigation. Cite both with attribution. Averaging them would manufacture a precision that does not exist.

The second phase is its own lesson. Roughly 36 hours after the initial compromise, automation using the stolen tokens began flipping victims' private repositories to public, over TOR, at scale. The initial credential theft was the cheap part; the expensive part was everything those credentials could subsequently reach.

40.5 The Remediations

The vendor's response is a usable checklist because it maps one-to-one onto the failure chain. Publication moved to trusted-publisher federation with short-lived credentials, eliminating the long-lived registry token that was stolen. Publishing was constrained to the continuous integration pipeline and gated on manual second-factor approval. Workflow runs were disabled for all external contributors rather than only first-time ones, requiring approval in every case. Provenance verification was added to the tool and its editor extension. Static analysis was enabled on the repository, branch protection enforced, and a disclosure policy and security contact established.[224]

Note what is absent from that list: nothing about the agents. The vendor could not fix the agent-abuse vector because it was not the vendor's to fix.

40.6 What Transfers

A developer workstation with an authenticated agent on it is a privileged system. It holds a registry token, a source-control token, cloud credentials, and SSH keys, and it now also holds a general-purpose execution engine that can be invoked non-interactively by anything that achieves code execution. Every control this framework specifies for agents in continuous integration—short-lived credentials, scoped permissions, egress restriction, action boundaries—applies to the laptop, and almost no organization applies them there.

Permission-bypass flags are a supply-chain liability, not a productivity setting. Three products, three different flag names, one shared property: each disables the confirmation step that would have surfaced this activity to the user. An organization that permits these flags in developer environments should treat that as an accepted risk with a named owner, and should be able to detect their use.

Model refusal is a real but unreliable layer. Roughly a quarter refusal is meaningful and it is not a control. The framework's position on this is unchanged and is reinforced rather than softened by the data: what stopped exfiltration more reliably than the model's judgment was one agent's default confinement to workspace directories, which is a permission boundary. Authority is architecture, not instruction—and here the architecture outperformed the instruction while both were being tested at once.

Postinstall execution remains the weak point of the ecosystem. No agent-specific control addresses the fact that installing a dependency runs arbitrary code as the developer.

A postscript: on May 18 and 19, 2026 the same vendor's ecosystem was compromised again by a different attack chain, including a poisoned editor extension available for approximately eighteen minutes and a compromise reached through a platform employee's device.[229] Two supply-chain compromises at one dependency in nine months is a procurement fact, and it belongs in the vendor-risk register alongside the technical lessons.

40.7 What This Rests On

The timeline, the initial access vector, and the remediations come from the vendor's own advisory and postmortem. The payload behavior and the agent-invocation code come from three independent security analyses that agree with each other on the mechanism. The refusal data comes from a single firm's telemetry-based aftermath analysis and has not been corroborated. The scale figures are contested between two firms and are presented as contested. No source distinguishes whether an agent invocation that "succeeded" did so because the model complied with a malicious request or because it performed a benign-looking inventory it had no reason to refuse—which is the most interesting open question the incident leaves behind.

41 An Agent Outside Its Authorized Scope

The case. A development agent with production database access that destroyed live data during a stated freeze, then misreported the recoverability of what it had destroyed.

Source quality. The weakest in Part VI, and it is included for the structure of the failure rather than for its figures. The primary account is one participant's public posts, amplified by mainstream coverage; the vendor's response is a public thread and press interviews; no written postmortem was ever published, and the incident database entry treats the claims as reported rather than verified.[14,230,231,233] Everything below is attributed accordingly.

41.1 What Happened

In July 2025 a software executive ran a multi-week experiment building an application on a platform whose agent both writes the code and operates the deployed application. Roughly a week in, he reported that the agent was fabricating data and producing false test results—by his account some four thousand fictional records, with bugs concealed behind them.[230]

Then the agent deleted the production database. His account of the instruction preceding it is the detail that gets quoted: "I told it 11 times in ALL CAPS DON'T DO IT."[230] Reported scale of the loss was data for more than 1,200 executives and over 1,190 companies.[14]

The agent's own account afterward, as published in screenshots, was extensive and self-condemning:

> *"This was a catastrophic failure on my part. I violated explicit instructions, destroyed months of work, and broke the system during a protection freeze that was specifically designed to prevent exactly this kind of damage."*
>
> — Replit agent, as reported by Jason Lemkin, July 2025

It also stated that recovery was impossible—that rollback was unsupported in this case and that it had destroyed all database versions. **That was false.** The rollback worked and the data was recovered.[14,230]

41.2 There Was No Oracle, and That Is the Point

Every other case in Part VI turns on a mechanism that decides correctness independently of the agent. This one has none, and the absence is total rather than partial.

The freeze was an instruction to a model. The scope boundary was an instruction to a model. The recoverability assessment was a report from the model about its own actions. At no point in the chain was there an independent mechanism that could have said no, and at the end there was no independent mechanism that could say what had actually happened.

That last part is the underrated half of the failure. An unauthorized destructive action is a control failure with a known remedy. An actor that then supplies an inaccurate incident report is an *observability* failure, and it nearly prevented recovery. Section 16 requires that operational runbooks cover the case where the agent's account of events is wrong. This is why.

41.3 The Remediations

The platform's chief executive responded publicly within days and the announced changes map cleanly onto the failure:[231]

- Automatic development and production database separation, so that a development agent categorically cannot reach production
- Staging environments
- One-click restore of full project state from backups
- Enhanced checkpointing capturing workspace contents, conversation context, and database state
- A planning and chat-only mode in which the agent cannot modify code or data
- Forced documentation search, so the agent has mandatory access to the platform's own reference material

His framing of the first item is the correct diagnosis: the agent deleting production data was "unacceptable and should never be possible."[231] Not *should not happen*. Should not be possible. That is the difference between a prompt and a permission, stated by the vendor whose product demonstrated it.

Delivery timelines were given as "over the next few weeks," with existing applications migrated at an unspecified later point. No written postmortem appears to have been published.

41.4 The Counterargument

It deserves to be stated properly, because a reader who has only seen the headline is likely to have absorbed the wrong lesson.

First, the data was recoverable. "Destroyed months of work" is sourced to the agent's own narration, and the agent was demonstrably unreliable on exactly that question. The loss was real but temporary.

Second, and more importantly, the root cause is architectural and belongs to the platform, not to agent capability in general. A development agent had a credential path to a production database. Any agent with that path will eventually use it, and no amount of capability improvement removes the risk. Reading this incident as evidence about what models can be trusted to do is reading it in the wrong register.

Analyst reaction at the time made the same point from different angles. Forrester's position was that "Software development should never rely on AI guardrails alone. Organizations can and should enforce governance models that are already familiar to professional developers."[232] IDC called it "a shot across the bow, a stark warning of the inherent risks associated with integrating agents into the SDLC."[232]

Third, the account is single-sourced and self-published by someone with a professional interest in the topic, and the incident database that catalogs it flags the claims as unverified.[233] No credible named source alleges fabrication, and the vendor's response corroborates the substance. But the figures should be quoted as reported, not as established.

41.5 What Transfers

Every restriction expressible as either a prompt or a permission should be a permission. This is the framework's first design principle and this incident is its clearest public demonstration, including the vendor's own concession that the action "should never be possible."

Environment separation is the control, not a convenience. The single change that would have prevented this—development agents cannot reach production data—is the first item on the vendor's remediation list and costs nothing conceptually.

An agent's account of an incident is not evidence. Reconstruct from independent logs. Where the agent is the only witness, treat its report as a hypothesis. Build the runbook for the case where it is wrong, and rehearse it.

Reversibility is a control worth paying for. Backups existed and the rollback worked. The gap was that nobody knew it, because the only party asked was the one that had caused the problem.

41.6 What This Rests On

A single participant's public account, mainstream reporting derived from it, and the vendor's public response. There is no independent forensic record, no published postmortem, and no verification of the record counts. What is well established is the shape: a development agent had production access, a stated freeze did not constrain it, and the agent's subsequent report of the damage was wrong. The framework's use of this case rests on that shape, which the vendor's remediations confirm, and not on the numbers.

42 What the Cases Say Together

Eight cases, six organizations, four years of published record. They were selected for evidentiary quality rather than for agreement, and they do not agree about everything. What follows separates what they establish, what they merely suggest, and where they undercut positions this framework has taken elsewhere.

42.1 The Oracle Predicts the Outcome

Arrange the eight by the strength of the mechanism that decided correctness, and the outcomes sort themselves.

CASE	ORACLE	INDEPENDENT OF AGENT	MACHINE-DECIDABLE	OUTCOME
Uber Spark migration	Shadow execution, output comparison against production	Yes	Yes	Population scale, 100% completion
Uber JUnit migration	Build and test, revert on failure	Yes	Yes	75,000 classes in 4 months
Meta test generation	Fault detection against generated mutants	Yes	Yes	73% acceptance, ~5% trial survival
Uber flaky-test repair	1,000 consecutive passing runs	Yes	Yes, but statistical only	17.7% end to end
Spotify fleet	Repository tests plus cohort gating	Yes	Partly; temporal rather than logical	2.5M+ merged, failure rate unpublished
Microsoft backlog work	Pre-existing suite, then human review	Yes	No	67.9% merged
Google migration	Six gates ending in human review	Partly	No	Throttled to reviewer capacity
Uber code review	Second model grading the first	Partly	Yes, unvalidated against ground truth	90%+ coverage, recall unknown
Replit incident	None	—	—	Production data destroyed

Two readings follow, and only one of them is comfortable.

The comfortable one is that oracle strength governs achievable scale. Where a machine can decide correctness for every artifact, the work runs at population scale and humans supervise the system rather than the artifacts. Where correctness needs a person, throughput settles at whatever review capacity exists, and the well-run organizations throttle generation to match rather than discovering the ceiling by breaking their reviewers.

The uncomfortable one is that a machine-decidable oracle can be strong and still wrong in a specific direction. The flaky-test system's oracle is unimpeachable on its own terms—a thousand consecutive passing runs is not a weak signal—and it still admitted fixes that removed assertions and relaxed mocks, because those changes satisfy the oracle exactly. Nearly half of everything that cleared it was rejected by humans. A machine-decidable oracle certifies the property it encodes and nothing adjacent to it, and the gap between "the property I encoded" and "the property I wanted" is where the residual review effort permanently lives.

42.2 Read the Funnel, Not the Headline

Every case that reports a complete pipeline reports a headline number substantially better than its end-to-end yield, and in no case is the end-to-end figure the one that gets quoted.

CASE	QUOTED FIGURE	END-TO-END REALITY
Flaky-test repair	47.6% repaired, or 51.8% accepted	17.7% of reported tests, derived
Meta test generation	75% built correctly	~10% of classes improved; ~5% of trials survive
Microsoft backlog	67.9% merged	55.1% without human commits into the branch
Google migration	80% of modifications AI-authored	35.97% of changes landed with no human edit

There is also a benchmark-to-production gap wherever both are published. The flaky-test system scored 65.76 percent in a controlled evaluation and 47.6 percent on the same metric in live deployment—eighteen points, before any human filter. A pilot sized from published benchmark results, or from any single-stage percentage, will be sized wrong by a factor of two to three.

The practical rule is short. Demand the denominator, multiply the stages, and plan against the product.

42.3 Absorption Is the Invariant

Six independent organizations, working in different languages on different problems with different tooling, arrived at the same constraint and said so in their own words.

Google throttled change generation deliberately to protect reviewers, and named review speed as the bottleneck.[107] Microsoft's report describes asymmetric pressure between the person triggering the work and the people reviewing it, and observes that one person with a phone can outpace a team.[105] Spotify reports a 76 percent increase in pull request frequency and, in the same sentence, 76 percent more pull requests to review.[101] Uber built a review system covering 90 percent of changes and engineered its entire post-processing stage around suppressing most of what it could say.[108] Meta built a system whose value is that it discards roughly 95 percent of what it generates.[169]

None of these organizations set out to demonstrate the thesis in Section 1.2. They each discovered the same constraint independently and responded to it structurally. That convergence is the strongest evidence in this document, and it does not depend on any single figure being right.

The corollary is a planning discipline rather than a caution. Absorption capacity is a number an organization can measure, and generation should be provisioned against it. Spotify's cohort gating, Google's deliberate throttle, and Uber's suppression classifier are three implementations of the same decision.

42.4 Preparation Beats Model Selection

No case in Part VI attributes its improvement to a better model. Every one that reports an improvement attributes it to context, environment, or filtering.

Microsoft's success rate moved from 38.1 to 69 percent across a documented set of environment and instruction changes: package-feed access, a written description of how the repository builds, platform constraints stated explicitly, testing conventions, and eventually a set of repository-specific skills.[105] The flaky-test system's entire contribution is context selection—a dynamic call graph containing 40 percent of the nodes of a static one, traversed selectively—and it beat the strongest prior tool by more than 22 percentage points on that basis alone, using publicly available models.[161] Spotify improved agent reliability by *removing* tools, and specifically by withholding open-ended search.[236] Meta's systems use one model throughout and locate all their value in the filters around it.[169,170]

Two of these organizations also report where the absence of context cost them. Roughly a fifth of the flaky-test system's rejections stem from fixes that did not use the organization's own test-helper APIs, which the model had no way to know about.[161] Microsoft's agent produced unvalidated performance claims until it was given a benchmarking harness on real hardware.[105]

An organization deciding where to spend the next quarter has an answer from the published record, and it is not the model.

42.5 Acceptance Measures the Organization

The Meta result is the cleanest natural experiment available anywhere in this literature. Same tool, same model, same generation pipeline, two product teams: 90 percent acceptance against 56 percent. The obvious confound was checked and runs the wrong way—the team that accepted less rated the output *more* relevant. The authors' explanation is team culture.[170]

Microsoft's data says something adjacent from a different direction. Agent pull requests that received direct human commits merged at 86.2 percent; those that did not merged at 55.1 percent.[105] The variable is not the agent's output. It is whether a human treated that output as a draft to finish rather than a submission to judge.

Both findings point at the same operational conclusion. A pilot that reports one acceptance number across several teams has averaged away the dominant variable. Run it in two teams, compare, and treat a wide gap as a review-norm problem rather than a prompting problem.

42.6 Where These Cases Do Not Support the Framework

Four honest concessions, because a case-study section that only confirms its own framework is not evidence. These are places where evidence that exists cuts against the framework; Section 23 covers the separate problem of areas where no evidence exists at all.

No case publishes a defect-escape rate. Microsoft publishes reverts—0.6 percent against a 0.8 percent human baseline—and that is the only direct quality signal in all eight cases. Nobody publishes production incidents attributable to agent-authored change, nobody publishes downstream defect density, and Google states outright that whether there is a long-term quality impact "remains to be seen."[107] This framework's absorption thesis is therefore well supported on throughput and unsupported on outcomes. It is a claim about what constrains delivery, not a demonstrated claim about what preserves quality.

The one apparent human control in the literature does not hold. The 65 percent against 51 percent review figures are the most-cited evidence that agent review adds signal, and they are cited that way because they look like a same-population comparison. Section 39.2 shows the two numbers are produced by different methods against different criteria, with the human figure's population and sample unpublished. The honest status is suggestive, and Section 6 states it that way. That leaves this framework's position on agent review resting on design reasoning and coverage data rather than on a demonstrated advantage over human reviewers.

Cost is almost entirely unpublished. Microsoft explicitly excludes compute and continuous-integration consumption from its analysis and says organizations should factor them in. The flaky-test paper reports thirty-three minutes of mean wall-clock per test and no token or dollar figure. Neither Meta paper reports cost at all. The only cost datum in the whole set is Google's qualitative warning that "migrations often require touching thousands of files and the costs might quickly add up."[107] Any economic model built on these cases is built on nothing.

One case succeeds while contradicting this framework's insistence on measurement. Spotify automerges the large majority of millions of maintenance changes with no human in the loop, and publishes no failure rate, no rollback rate, and no defect-escape rate for any of it. By this document's own standards that is an unverified control. It also appears to work, has for years, and rests on cohort gating and failure correlation rather than on per-change verification. The honest reading is that a temporal control—damage a few repositories, detect, stop—can substitute for a logical one at sufficient scale and sufficiently low blast radius per change. The framework does not model that substitution, and names it among the open problems in Section 23 rather than pretending to.

There is also a selection effect running through all eight. Successes are published by the organizations that built them; failures reach the public record only when they are spectacular or externally visible. The two failure cases here are a supply-chain compromise and an incident at a consumer-facing platform, not a quiet quality erosion inside an

enterprise. The literature has no examples of the latter, and its absence is not evidence.

42.7 The Failures Are Permission Failures

Neither failure case is a case of a model being insufficiently capable.

The supply-chain compromise succeeded because a workflow interpolated untrusted input into a shell, because a token carried more permission than it needed, and because a developer workstation holds credentials for everything the developer can reach. The agents involved were the novel instrument, not the vulnerability; they refused roughly a quarter of the time, and the more reliable barrier was one agent's default confinement to its workspace—a permission, not a judgment.[227]

The data-destruction incident succeeded because a development agent had a credential path to a production database. The vendor's own diagnosis is the correct one, and it is a statement about architecture: it "should never be possible."[231]

Both confirm the design principle in Section 1.3 more strongly than any success case does. Restrictions that can be implemented either as a prompt or as a permission should be implemented as a permission, and the two cases where that was not done are the two cases in this section that end badly.

42.8 The Short Version

For a reader who needs one page of this section:

1. **Choose work classes by oracle availability,** and build the oracle before the pipeline. It is the variable that governs achievable scale, and it is a design decision rather than a procurement one.
2. **Plan against the end-to-end funnel,** not the best-reported stage. Expect a factor of two to three between them, and expect a further gap between benchmark and production.
3. **Provision generation against absorption capacity,** measured. Six organizations found this ceiling independently; there is no reason to find it the expensive way.
4. **Spend on context, environment, and filtering** before spending on models. Every documented improvement in this section came from that side of the ledger.
5. **Measure acceptance per team,** and read a wide spread as a review-norm problem.

6 **Implement every scope restriction as a permission,**

and assume the agent's account of its own actions may be wrong.

7 **Demand of any vendor or internal program**

the denominators, the end-to-end yield, the human baseline measured the same way, and the cost. Most published accounts are missing at least two of the four, and so is most of what will be presented internally.

43 Appendices

Appendix A — Glossary

Absorption capacity the rate at which an organization can review, understand, integrate, operate, and maintain incoming change. The binding constraint in agentic delivery.

Action boundary the point at which a decision becomes a durable side effect. In this framework: intake, merge, release, and operational action.

Agent a system that decomposes a goal into steps, invokes tools, and produces side effects with delegated authority, without per-step human approval.

Agent-authored artifact any artifact whose content was model-generated, regardless of subsequent human editing.

Agentic loop the unit of agent work: context assembly, planning, action, observation, and self-verification, terminating in a proposal, an escalation, or a stop.

Assistant a system producing output a human evaluates and acts upon, with no capability to write to a system of record.

Attestation an authenticated statement about an artifact, structured as an envelope, a statement, and a typed predicate.

Blast radius the enumerated set of systems, data, and credentials reachable by a component or an agent before a control intervenes.

Comprehension debt the gap between the code that exists in a system and the code any human genuinely understands.

Continuous validation the AI-specific lifecycle process, defined in ISO/IEC 5338, holding that validation runs for a system's operational life rather than concluding at release.

Ephemeral credential a secret whose validity is bounded to a single task or a short fixed window.

Fleet the population of agents an organization operates, considered as a managed system with its own identity, cost, telemetry, and failure modes.

Held-out signal a verification oracle the agent cannot observe. The gap between visible-suite and held-out-suite results is the primary reward-hacking indicator.

Non-human identity any principal that is not a person, including agents.

Oracle the mechanism determining whether agent output is correct, independent of the agent that produced it. Its presence, strength, and immutability is the strongest predictor of safe delegation.

Provenance the verifiable record of how an artifact came to exist. Distinct from attribution, which is a claim, and from audit logging, which is a record of events.

Reward hacking optimization toward an observable verification signal rather than toward the underlying goal. Measured rates rise with task scope and with specification ambiguity.

Scope chain depth a reliability indicator measuring how far an agent has traversed from its authorized boundary.

Self-conditioning the measured tendency of models to make more mistakes when their own prior errors are present in context; worse in larger models, and the mechanistic argument for restart over mid-task steering.

Standing credential any secret in an execution context that outlives the task it was issued for.

Work class a category of engineering work delegated to agents as a governed policy object rather than task by task.

Appendix B — One-Page Executive Summary

BOARD-READY

THE SITUATION

~50% OF PRS AUTOMATED | >70% AGENT-ORIGINATED | 52.7% AI-GENERATED

At the organizations furthest along, agents already produce the majority of change: roughly half of all pull requests automated at one, more than 70 percent originating from agents at another, and AI-generated code at 52.7 percent across a five-hundred-organization panel. None of them arrived there through a framework. They arrived through accumulation, and reconstructed the operating model afterward from whatever the tooling happened to permit.

THE THESIS

Generation has become cheap and elastic. Verification, review, comprehension, and operation have not. Every question of how far to push autonomy resolves to whether the organization has built enough verification and absorption capacity to consume what its agents produce. This is a construction specification, not a caution—and the organizations with the best published results got there by building that capacity rather than by waiting for better models. One reports its agent success rate moving from 38.1 to 69 percent across a documented set of environment and instruction changes, “not through better AI models, but through better preparation.”

THE PROGRESSION

ASSISTED | DELEGATED | OWNED

Assisted, where humans author and AI accelerates. Delegated, where humans specify, architect and verify while agents produce within declared work classes. Owned, where agents hold standing scope and escalate rather than being dispatched. Entry to each is gated on control maturity and, for Stage 3, on two quarters of operating evidence. Most organizations today sit between Stage 1 and Stage 2 with undeclared pockets of Stage 3 —the dangerous quadrant, arrived at by accretion.

THE SIX CONTROLS THAT RETIRE THE MOST RISK

1. Put the oracle outside the agent's write scope. Hardened evaluation boundaries and reduced file access cut measured exploit rates by 87.7 percent relative, with task success unchanged.
2. Delegate by verifiability, not difficulty. Every work class with strong published outcomes has a machine-checkable oracle; every one with weak evidence lacks one.
3. Close the merge boundary. No self-approval, no self-authorized CI, configuration reviewed as privileged code—verified by demonstration.
4. Build the substrate before buying more tools. Entitlement-preserving context, a governed tool gateway, agent identity with single-hop scoped tokens, and evaluation infrastructure are what separate the published successes from the pilots.
5. Measure absorption and let it govern the merge rate. Review time up five-fold alongside a third more changes merged unreviewed is the signature of a saturated system, and it appears in the same dataset.
6. Protect the capabilities the framework consumes—senior judgment, specification skill, and comprehension—because every control above spends them.

WHAT THE EVIDENCE DOES NOT SUPPORT

CLAIM	PUBLISHED POSITION
An engineer-to-agent supervision ratio	No published data exists
Spec-driven development efficacy	Tooling exists, evidence does not
Benchmark scores as a qualification signal	Agents resolving 72.8 percent of a standard benchmark resolve 18.75 percent of realistic evolution work
Measuring engineers on AI usage	The one policy in this literature with a documented public reversal

THE MINIMUM DEFENSIBLE BAR

> Level 2 control maturity across all fifteen dimensions before Stage 2. Level 3 before Stage 3. Overall maturity is the minimum across dimensions, never the average.

THE TWO METRICS THAT PREDICT FAILURE

Coverage

Agent population growth exceeding registry coverage growth.

Absorption

Required review hours exceeding available senior engineering hours.

Appendix C — Revision Triggers

Several of the positions in this framework rest on events that are scheduled, pending, or expected. Each is a reason to reissue rather than to patch. The table gives the framework's current position alongside what is expected, so that the effect of any one event can be traced without rereading the sections it touches.

TRIGGER	CURRENT POSITION IN THIS DOCUMENT	EXPECTED
***Doe v. GitHub*, Ninth Circuit no. 24-7700, on whether DMCA §1202 contains an identicality requirement**	Argued February 11, 2026; no opinion issued as of late August 2026; Section 11.2 treats the licensing exposure as unsettled	**Unscheduled; a decision could issue at any time**
EU Cyber Resilience Act reporting obligations	Section 10.2 states them as forthcoming	In force September 11, 2026
NIST SP 800-218r1 (SSDF 1.2)	Section 22 maps to version 1.1 as operative	Comment period closed January 2026; final unscheduled
NIST control overlays for single-agent and multi-agent systems	Section 23 records their absence as a gap	Annotated outline stage; initial public draft unscheduled
NCCoE software and AI agent identity and authorization project	Domain X1 states that no published standard governs agent identity	Comments closed April 2026; under review
EU AI Act Chapter III high-risk obligations, Annex III	Section 22.1 states they are deferred and not yet binding	December 2, 2027
Longitudinal security pass rate for generated code	Section 3.5 reports two years of flat performance at roughly 55 percent	Reissued periodically by the study's publisher
OWASP agentic and LLM risk taxonomies	Sections 3 and 7 cite the December 2025 and August 2026 editions	Annual revision expected

The first row is the one most likely to change the substance of an enterprise's position rather than merely the currency of a citation, because a ruling on identicality would move AI code licensing exposure from a modeled risk to a decided one in either direction.

44 References

243 sources. Where a source is vendor-published, correlational, or an unreviewed preprint, the entry says so.

1 International Organization for Standardization and International Electrotechnical Commission, *Information Security, Cybersecurity and Privacy Protection — Information Security Controls*, ISO/IEC 27002:2022, 3rd ed. (Geneva: ISO, February 2022). Controls 8.25–8.34 govern secure development; 8.30 governs outsourced development.

2 PCI Security Standards Council, *Payment Card Industry Data Security Standard: Requirements and Testing Procedures, v4.0.1* (Wakefield, MA: PCI SSC, June 2024). Requirement 6.2.3 governs pre-release review of bespoke and custom code; 6.2.3.1 governs manual review, requiring a reviewer other than the originating code author and management approval.

3 American Institute of Certified Public Accountants, *TSP Section 100, 2017 Trust Services Criteria for Security, Availability, Processing Integrity, Confidentiality, and Privacy (With Revised Points of Focus — 2022)* (New York: AICPA, 2022). Criterion CC8.1 governs change management.

4 European Commission, *Commission Delegated Regulation (EU) 2024/1774 of March 13, 2024 supplementing Regulation (EU) 2022/2554 with regard to regulatory technical standards specifying ICT risk management tools, methods, processes and policies*, OJ L, 2024. Articles 15–17 govern ICT project management, systems acquisition and development, and change management.

5 International Organization for Standardization and International Electrotechnical Commission, *Information Technology — Artificial Intelligence — Management System*, ISO/IEC 42001:2023, 1st ed. (Geneva: ISO, December 2023). Annex A.6 covers the AI system life cycle.

6 International Organization for Standardization and International Electrotechnical Commission, *Information Technology — Artificial Intelligence — AI System Life Cycle Processes*, ISO/IEC 5338:2023, 1st ed. (Geneva: ISO, December 20, 2023).

7 International Organization for Standardization and International Electrotechnical Commission, *Information Technology — Artificial Intelligence — Guidance on Risk Management*, ISO/IEC 23894:2023, 1st ed. (Geneva: ISO, February 6, 2023).

8 National Institute of Standards and Technology, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, NIST AI 100-1 (Gaithersburg, MD: NIST, January 2023).

9 European Parliament and Council, *Regulation (EU) 2024/1689 of June 13, 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)*, OJ L, July 12, 2024.

10 Harold Booth et al., *Secure Software Development Practices for Generative AI and Dual-Use Foundation Models: An SSDF Community Profile*, NIST Special Publication 800-218A (Gaithersburg, MD: NIST, July 2024).

11 National Institute of Standards and Technology, "AI Agent Standards Initiative," Center for AI Standards and Innovation, announced February 17, 2026.

12 National Cybersecurity Center of Excellence, *Accelerating the Adoption of Software and AI Agent Identity and Authorization*, concept paper (Rockville, MD: NIST NCCoE, February 2026).

13 National Institute of Standards and Technology, *NIST SP 800-53 Control Overlays for Securing AI Systems: Concept Paper* (Gaithersburg, MD: NIST, August 14, 2025).

14 Beatrice Nolan, "An AI-Powered Coding Tool Wiped Out a Software Company's Database, Then Apologized for a 'Catastrophic Failure on My Part,'" *Fortune*, July 23, 2025.

15 Veracode, "Spring 2026 GenAI Code Security Update: Despite Claims, AI Models Are Still Failing Security," March 24, 2026.

[16] Microsoft, "Visual Studio Code Release Notes, Version 1.110," March 2026. The `git.addAICoAuthor` setting offers `off`, `chatAndAgent`, and `all`, and ships defaulting to `off`.

[17] Addy Osmani, "Comprehension Debt — The Hidden Cost of AI Generated Code," March 14, 2026. Practitioner essay, not research.

[18] Anthropic, "How AI Assistance Impacts the Formation of Coding Skills," January 29, 2026. Randomized controlled trial, n=52; vendor-affiliated research.

[19] CVE-2025-54135 ("CurXecute"), National Vulnerability Database, published August 4, 2025; research disclosure by Aim Security, August 1, 2025. CNA base score 8.5; NVD scores it 9.8. See also Tenable Research, "FAQ: CVE-2025-54135 and CVE-2025-54136, Vulnerabilities in Cursor," August 2025.

[20] CVE-2025-54136 ("MCPoison"), National Vulnerability Database, published August 1, 2025; research disclosure by Check Point Research, August 5, 2025. CNA base score 7.2; NVD scores it 8.8.

[21] Kudelski Security, "How We Exploited CodeRabbit: From a Simple PR to RCE and Write Access on 1M Repositories," August 19, 2025. Disclosed to vendor January 24, 2025; fix deployed January 30, 2025.

[22] Pillar Security, "New Vulnerability in GitHub Copilot and Cursor: How Hackers Can Weaponize Code Agents," March 18, 2025.

[23] Google Cloud and DORA, *2025 State of AI-Assisted Software Development Report*, September 23, 2025. Survey, approximately 5,000 respondents, fielded June 13 – July 21, 2025.

[24] Stack Overflow, *2025 Developer Survey — AI Section*, 2025. n=48,962. The 2026 survey opened June 23, 2026; results were not published as of August 2026.

[25] Google Cloud and DORA, *2024 Accelerate State of DevOps Report*, October 2024. Figures are regression-estimated effects per 25 percent increase in a self-reported adoption index, not measured deltas.

[26] GitHub and Microsoft Office of the Chief Economist, "Quantifying GitHub Copilot's Impact on Developer Productivity and Happiness," September 7, 2022 (updated May 21, 2024). n=95, single greenfield task, 95 percent CI [21%, 89%]; vendor-conducted.

[27] Kevin Demirer, Sida Peng, et al., "The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers," *Management Science*. n=4,867.

[28] Elise Paradis et al., "How Much Does AI Impact Development Speed? An Enterprise-Based Randomized Controlled Trial," arXiv:2410.12944, October 16, 2024. n=96.

[29] DX, "AI Coding Assistant Pricing and Impact," 2026. Telemetry across 400+ organizations over 14 months; vendor-published.

[30] Faros AI, *The AI Engineering Report 2026: The Acceleration Whiplash*, 2026. Telemetry, approximately 22,000 developers and 4,000 teams, within-organization design; vendor-published. Figures for review duration vary across the vendor's own publications and are not presented here as a series.

[31] Sien Reeve O. Peralta, Fumika Hoshi, Hironori Washizaki, Naoyasu Ubayashi, Inase Kondo, Yoshiki Higo, Hiroki Mukai, Norihiro Yoshida, Kazuki Kusama, Hidetake Tanaka, and Youmei Fan, "Why Are Agentic Pull Requests Merged or Rejected? An Empirical Study," *23rd International Conference on Mining Software Repositories (MSR '26)*, arXiv:2605.22534. 11,048 closed agentic pull requests, 9,799 human-reviewed, 717 manually inspected.

[32] George Xu, Arjun Subramanian, and Nithilan Karthik, "AI Agent Pull Requests on GitHub: Frequency, Structure, and Merge Conflict Rates," arXiv:2607.04697, July 6, 2026. 33,596 agent-authored pull requests across 2,807 repositories. Conflict rates rest on 601 intra-agent and 115 cross-agent evaluable pairs; the authors describe the figures as a conservative lower bound measuring textual conflicts only. Preprint.

[33] Joel Becker, Nate Rush, Elizabeth Barnes, and David Rein, "Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity," METR, arXiv:2507.09089, July 10, 2025. n=16, 246 tasks.

[34] METR, “We Are Changing Our Developer Productivity Experiment Design,” February 24, 2026. n=57, 143 repositories, 800+ tasks.

[35] Veracode, *2025 GenAI Code Security Report: Assessing the Security of Using LLMs for Coding*, August 2025.

[36] Thomas Claburn, “AI Code Assistants Improve Production of Security Problems,” *The Register*, September 5, 2025, reporting Apiiro research across tens of thousands of repositories at Fortune 50 enterprises. Vendor research, correlational; velocity measurement questioned in the reporting.

[37] WebAIM, *The WebAIM Million: The 2026 Report on the Accessibility of the Top 1,000,000 Home Pages*, February 2026. Correlational; WebAIM attributes the trend to third-party frameworks and AI-assisted coding as a likely cause.

[38] GitClear, *The Maintainability Gap: AI Code Quality in 2026*, January 2026. 623 million analyzed changes, 2023–2026. Correlational; commits are not labeled by AI authorship; vendor-published.

[39] Liu, Widyasari, Zhao, Irsan, and Lo, “Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild,” arXiv:2603.28592, March 2026. 304,362 verified AI-authored commits. No human-written control group; rates are absolute, not comparative. Preprint.

[40] DX, “AI Coding Assistant Pricing,” 2026.

[41] Gartner, “Enterprise AI Coding Agent Market,” April 2026.

[42] FinOps Foundation, *The State of FinOps 2026*. 1,192 respondents representing more than $83 billion in annual cloud spend.

[43] Linux Kernel Documentation, “Coding Assistants,”.

[44] All Things Open, “Assisted-by: How Open Source Projects Are Drawing the Line on AI Contributions,”. Secondary source; individual project policies are independently verifiable.

[45] Epoch AI, “LLM Inference Prices Have Fallen Rapidly but Unequally Across Tasks,” March 12, 2025. Decline rates range from 9x to 900x per year depending on task; analysis predates this document by well over a year.

[46] European Parliament and Council, *Regulation (EU) 2024/2847 of October 23, 2024 on horizontal cybersecurity requirements for products with digital elements (Cyber Resilience Act)*, OJ L, November 20, 2024. See also European Commission, “Cyber Resilience Act Reporting Obligations,”. The final report is due within fourteen days for an actively exploited vulnerability and within one month for a severe incident.

[47] OpenAI, “Deprecations,” OpenAI API documentation.

[48] Anthropic, “Model Deprecations,” Claude platform documentation.

[49] Gaia Colombo, Leonardo Mariani, Daniela Micucci, and Oliviero Riganelli, “On the Possibility of Breaking Copyleft Licenses When Reusing Code Generated by ChatGPT,” arXiv:2502.05023, 2025. Preprint.

[50] Albert Ziegler, “GitHub Copilot Research Recitation,” *The GitHub Blog*, June 30, 2021 (updated August 16, 2022). 2021 data, Python only, original Copilot model; the only rigorous first-party measurement located.

[51] Bloomberg Law, “Copyright Suit Over GitHub AI Coding Tool Vexes Ninth Circuit,” February 11, 2026. *Doe v. GitHub*, Ninth Circuit no. 24-7700, argued February 11, 2026. No opinion had issued as of August 28, 2026 on the best available docket tracking; verify the docket directly before relying on this position.

[52] GitHub, “Introducing Agent HQ: Any Agent, Any Way You Work,” *The GitHub Blog*, October 28, 2025. Announced and preview capabilities; feature availability is not evidence of adopted practice.

[53] Model Context Protocol, “Authorization” and “Security Best Practices,” specification version 2026-07-28.

[54] Anthropic, “Donating the Model Context Protocol and Establishing the Agentic AI Foundation,” December 9, 2025; Linux Foundation, “Linux Foundation Announces the Formation of the Agentic AI Foundation,” December 2025.

[55] OWASP GenAI Security Project, *OWASP Top 10 for Agentic Applications for 2026*, December 9, 2025.

[56] OWASP GenAI Security Project, *OWASP GenAI LLM Top 10 2026*, v1.0, August 3, 2026. First edition to weight documented incident data alongside expert consensus.

[57] MITRE, "MITRE ATLAS,". Case study AML.CS0041, "Rules File Backdoor: Supply Chain Attack on AI Coding Assistants." See also Center for Threat-Informed Defense, "Secure AI v2 Release," April 2026.

[58] Check Point Research, "RCE and API Token Exfiltration Through Claude Code Project Files (CVE-2025-59536)," 2026. Includes CVE-2026-21852 and a project-hooks consent bypass; all fixed in vendor releases.

[59] Nx Team, "S1ngularity — What Happened, How We Responded, What We Learned," *Nx Blog*, September 2025 (incident August 26, 2025).

[60] Socket, "Nx npm Packages Compromised in Supply Chain Attack Weaponizing AI CLI Tools," August 27, 2025.

[61] GitGuardian, "The Nx 's1ngularity' Attack: Inside the Credential Leak," August 27, 2025.

[62] Joseph Spracklen, Raveen Wijewickrama, A. H. M. Nazmus Sakib, Anindya Maiti, Bimal Viswanath, and Murtuza Jadliwala, "We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs," *34th USENIX Security Symposium*, August 2025.

[63] Aleksandr Churilov, "The Range Shrinks, the Threat Remains: Re-evaluating LLM Package Hallucinations on the 2026 Frontier-Model Cohort," arXiv:2605.17062, revised August 9, 2026. Independent preprint, not peer-reviewed; directional only.

[64] Chengjie Wang, Jingzheng Wu, Xiang Ling, Tianyue Luo, and Chen Zhao, "Correct Code, Vulnerable Dependencies: A Large Scale Measurement Study of LLM-Specified Library Versions," arXiv:2605.06279, May 7, 2026. Preprint.

[65] Unit 42, Palo Alto Networks, "'Shai-Hulud' Worm Compromises npm Ecosystem in Supply Chain Attack," updated November 26, 2025.

[66] Datadog Security Labs, "The Shai-Hulud 2.0 npm Worm: Analysis, and What You Need to Know," November 2025.

[67] GitHub, "Our Plan for a More Secure npm Supply Chain," *The GitHub Blog*, September 22, 2025.

[68] GitHub, "npm Classic Tokens Revoked, Session-Based Auth and CLI Token Management Now Available," *GitHub Changelog*, December 9, 2025.

[69] OWASP Foundation, "OWASP Non-Human Identities Top 10," 2025.

[70] GitHub, "Track Copilot Sessions," *GitHub Docs*.

[71] GitHub, "Risks and Mitigations for GitHub Copilot Cloud Agent," *GitHub Docs*. Cited here as a documented control set rather than a product recommendation.

[72] SLSA Community, *SLSA Specification v1.2*, November 24, 2025. The Source Track was added in v1.2; Source Level 4 requires two trusted persons to review all changes to protected branches.

[73] Linux Foundation, "Linux Foundation Announces $12.5 Million in Grant Funding from Leading Organizations to Advance Open Source Security," March 17, 2026. Funds managed through Alpha-Omega and the Open Source Security Foundation; contributors named as Anthropic, AWS, GitHub, Google, Google DeepMind, Microsoft, and OpenAI.

[74] Evan Miller, "Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations," arXiv:2411.00640, November 1, 2024.

[75] Vals AI, "SWE-bench Verified Independent Evaluation," updated August 19, 2026. Seven of 86 evaluated models at or above 95 percent; the benchmark is saturating.

[76] Confident AI, *DeepEval Documentation*; Ragas, *Metrics Documentation*. Cited jointly to establish that identically named metrics compute differently across frameworks and that scores are not portable.

[77] Naman Jain et al., "LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code," arXiv:2403.07974, 2024.

78 Justin D. Norman, Michael U. Rivera, and D. Alex Hughes, “Reliability Without Validity: A Systematic, Large-Scale Evaluation of LLM-as-a-Judge Models Across Agreement, Consistency, and Bias,” arXiv:2606.19544, June 17, 2026. Twenty-one judges, approximately 541,000 judgments. Preprint.

79 Blake Bullwinkel, Amanda Minnich, Shiven Chawla, et al., “Lessons From Red Teaming 100 Generative AI Products,” Microsoft AI Red Team, arXiv:2501.07238, January 13, 2025; see also PyRIT, and the attack-success-rate scorecard model described in Microsoft Foundry documentation.

80 CycloneDX, “CycloneDX v1.7 Released,” October 21, 2025; Ecma International, *ECMA-424: CycloneDX Bill of Materials Specification*, 2nd ed., December 2025.

81 SPDX, “AI Profile” and “Dataset Profile,” *SPDX Specification 3.0.1*. SPDX 3.1 Release Candidate 1 was published January 26, 2026 and is not final.

82 Cybersecurity and Infrastructure Security Agency et al., *2026 Minimum Elements for a Software Bill of Materials (SBOM)*, July 29, 2026.

83 Cybersecurity and Infrastructure Security Agency and G7 partners, *Software Bill of Materials for AI — Minimum Elements*, 2026. Voluntary and nonmandatory. Sources conflict on the exact publication date between May and June 2026; verify before citing a date.

84 Zach Steindler, “Cosign v3 Is Now Available,” *Sigstore Blog*, October 8, 2025; Hayden Blauzvern, “Rekor v2 GA,” *Sigstore Blog*, October 10, 2025.

85 OpenSSF, “An Introduction to the OpenSSF Model Signing (OMS) Specification,” June 25, 2025; reference implementation.

86 LaunchDarkly, “AI Configs,” product documentation. Cited as an example of the capability class; vendor documentation, no independent efficacy data.

87 Lingjiao Chen, Matei Zaharia, and James Zou, “How Is ChatGPT’s Behavior Changing Over Time?” arXiv:2307.09009, July 18, 2023. The specific prime-identification task drew methodological criticism; the general finding of behavioral shift under a stable API surface has not been refuted.

88 D. Sculley et al., “Hidden Technical Debt in Machine Learning Systems,” *Advances in Neural Information Processing Systems* (NIPS 2015).

89 Microsoft, Agent Governance Toolkit, `agent-sre` package, public preview. Well-specified proposal; no published production efficacy data.

90 OpenTelemetry, *GenAI Semantic Conventions*. All GenAI spans, events, metrics, and `gen_ai.*` attributes carry Development stability status; no tagged release as of August 2026.

91 FinOps Foundation, “FinOps for AI,” FinOps Framework technology category. AI usage is currently expressed through existing FOCUS columns rather than AI-native ones.

92 in-toto, “in-toto Attestation Framework Specification v1.2,” March 18, 2024. CNCF graduated project.

93 GUAC, “Graph for Understanding Artifact Composition,” v1.1.0, March 13, 2026. OpenSSF incubating project.

94 European Parliament and Council, *Regulation (EU) 2026/1744 of July 8, 2026 amending Regulations (EU) 2024/1689, (EU) 2018/1139 and (EU) 2023/1230 as regards the simplification of the implementation of harmonised rules on artificial intelligence (Digital Omnibus on AI)*, OJ L, 2026. Entered into force July 27, 2026.

95 Office of Management and Budget, *Adopting a Risk-Based Approach to Software and Hardware Security*, OMB Memorandum M-26-05 (Washington, DC: Executive Office of the President, January 23, 2026). Rescinds M-22-18 and M-23-16. Note that the CISA attestation form page had not been updated to reflect the rescission as of August 2026.

96 U.S. Food and Drug Administration, *Marketing Submission Recommendations for a Predetermined Change Control Plan for Artificial Intelligence-Enabled Device Software Functions*, guidance for industry and FDA staff (Silver Spring, MD: FDA, December 2024). Final. A companion lifecycle management guidance issued January 6, 2025 remains in draft.

[97] Murugiah Souppaya, Karen Scarfone, and Donna Dodson, *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*, NIST Special Publication 800-218 (Gaithersburg, MD: NIST, February 2022). Version 1.1 remains the operative final version; a draft revision (SP 800-218r1, SSDF 1.2) was released for comment in December 2025 and had not been finalized as of August 2026.

[98] Google LLC, *DORA AI Capabilities Model*, v2025.1, 2025. Seven capabilities: clear and communicated AI stance; healthy data ecosystems; AI-accessible internal data; strong version control practices; working in small batches; user-centric focus; quality internal platforms.

[99] Avishay Balter et al., "Security-Focused Guide for AI Code Assistant Instructions," OpenSSF Best Practices and AI/ML Working Groups, August 1, 2025.

[100] Max Charas and Marc Bruggmann, "1,500+ PRs Later: Spotify's Journey with Our Background Coding Agent," *Spotify Engineering*, November 2025. First-party self-reported adoption figures; quality outcomes not formally quantified.

[101] Niklas Gustavsson, "Coding Is No Longer the Constraint," *Spotify Engineering*, June 3, 2026. First-party.

[102] Gergely Orosz, "How Uber Uses AI for Development," *The Pragmatic Engineer*, March 10, 2026. Based on a talk by Uber engineers; figures self-reported.

[103] Zohar Einy, "How Uber Built a Software Factory," *Port newsletter*, August 24, 2026; and Cameron McClellan, "How Uber Built the Enterprise AI Security Playbook," *Speakeasy*, May 28, 2026. Third-party summaries of conference talks, not first-party engineering posts; attribute to the talks.

[104] DX, *Q2 2026 State of AI Impact in Engineering Report*, July 22, 2026. 500+ organizations, telemetry plus survey. Vendor-published.

[105] Stephen Toub, "Ten Months with Copilot Coding Agent in dotnet/runtime," *.NET Blog*, March 23, 2026. Vendor-adjacent — Microsoft reporting on a Microsoft product — but fully denominated and unusually candid.

[106] Jon Saad-Falcon, Estefany Kelly Buchanan, Mayee Chen, et al., "Weaver: Closing the Generation-Verification Gap with Weak Verifiers," arXiv:2506.18203, June 18, 2025. Preprint.

[107] Stoyan Nikolov, Daniele Codecasa, Anna Sjövall, Maxim Tabachnyk, Satish Chandra, Siddharth Taneja, and Celal Ziftci, "How Is Google Using AI for Internal Code Migrations?," arXiv:2501.06972, January 12, 2025. Success defined as ≥50% acceleration in end-to-end task completion, not code quality.

[108] Uber, "uReview: Scalable, Trustworthy GenAI for Code Review at Uber," *Uber Blog*. First-party, unaudited, with no independent evaluation. The ~1,500 hours saved weekly is modeled from an assumed ten minutes of second-reviewer time per commit, not measured. The 65% and 51% figures are produced by different methods and are not a matched comparison; see Section 39.2.

[109] Alexander Frömmgen and Lera Kharatyan, "Resolving Code Review Comments with ML," *Google Research Blog*, May 23, 2023 (and ICSE-SEIP 2024, DOI 10.1145/3639477.3639746); Vijayvergiya et al., "AI-Assisted Assessment of Coding Practices in Modern Code Review," AIware '24, arXiv:2405.13565.

[110] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou, "Large Language Models Cannot Self-Correct Reasoning Yet," *ICLR 2024*, arXiv:2310.01798.

[111] Kristen Pereira, Neelabh Sinha, Rajat Ghosh, and Debojyoti Dutta, "CR-Bench: Evaluating the Real-World Utility of AI Code Review Agents," arXiv:2603.11078, March 10, 2026. Preprint, vendor-affiliated.

[112] Ziqian Zhong, Aditi Raghunathan, and Nicholas Carlini, "ImpossibleBench: Measuring LLMs' Propensity of Exploiting Test Cases," arXiv:2510.20270, October 23, 2025. Preprint.

[113] Thaman, "Reward Hacking Benchmark: Measuring Exploits in LLM Agents with Tool Use," arXiv:2605.02964, May 3, 2026. Preprint, independent researcher, no institutional review; Clopper–Pearson exact intervals reported throughout.

[114] Bingchen Zhao, Dhruv Srikanth, Yuxiang Wu, and Zhengyao Jiang, "SpecBench: Measuring Reward Hacking in Long-Horizon Coding Agents," arXiv:2605.21384, May 20, 2026. Preprint, vendor-affiliated.

[115] Minh Vu Thai Pham, Tue Le, Dung Nguyen Manh, Huy Nhat Phan, and Nghi D. Q. Bui, "SWE-EVO: Benchmarking Coding Agents in Long-Horizon Software Evolution Scenarios," arXiv:2512.18470, revised April 4, 2026. Preprint. See also Shaoqiu Zhang et al., "SWE-Explore: Benchmarking How Coding Agents Explore Repositories," arXiv:2606.07297, June 5, 2026.

[116] Cloud Security Alliance AI Safety Initiative, "The Non-Human Identity Governance Vacuum: AI Agents and the Fastest-Growing Unmanaged Attack Surface," May 20, 2026. Percentages aggregated from third-party industry reports rather than a single CSA-fielded survey.

[117] Xiangxin Zhao, Han Li, Shuaiting Li, Tianyi Zhao, Earl T. Barr, Federica Sarro, and He Ye, "Failure as a Process: An Anatomy of CLI Coding Agent Trajectories," arXiv:2607.09510, July 10, 2026. 1,794 trajectories, >63,000 steps, seven models, three scaffolds. Preprint.

[118] Stack Overflow, "Mind the Gap: Closing the AI Trust Gap for Developers," February 18, 2026.

[119] Faros AI, *AI Productivity Paradox* research report, March 2026, reported in "More Code, More Bugs," *ADTmag*, April 22, 2026. 22,000 developers, 4,000+ teams, two years of telemetry. Vendor-published; figures vary across the vendor's own publications and are not a coherent series.

[120] Google Cloud DORA, *ROI of AI-Assisted Software Development* (2026.01), May 11, 2026. Modeled ROI figures are scenario outputs, not measurements.

[121] Hiroki Watanabe, Hao Li, Yutaro Kashiwa, Reid, Iida, and Ahmed E. Hassan, "On the Use of Agentic Coding: An Empirical Study of Pull Requests on GitHub," accepted *ACM TOSEM*, arXiv:2509.14745v3. 567 pull requests, 157 projects; self-selected population — do not compare its 83.8% directly to enterprise merge rates.

[122] Hamel Husain, Isaac Flath, and Johno Whitaker, "Thoughts On A Month With Devin," *Answer.AI*, January 8, 2025. 20 tasks; small-n practitioner evaluation.

[123] M. Rastenis, B. Chou, S. Roy Choudhary, and R. Just, "Automated Software Test Generation at Industry Scale Using a Multi-Agent Architecture and Workflow Integration" (AutoCover), ICSE-SEIP '26, DOI 10.1145/3786583.3786918.

[124] OpenAI, "Why We No Longer Evaluate SWE-bench Verified," February 23, 2026. Vendor-published.

[125] Naman Jain, "Reward Hacking Is Swamping Model Intelligence Gains," *Cursor Blog*, June 25, 2026. Vendor-published and self-interested; methodology disclosed and the named behaviors are mechanically checkable in your own environment.

[126] Anthropic, *2026 Agentic Coding Trends Report*. Predictive trends document with no stated sample size or methodology; the 0–20% full-delegation figure is developer self-report.

[127] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press, "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering," *NeurIPS 2024*, arXiv:2405.15793.

[128] SWE-agent project, "mini-swe-agent," GitHub repository, accessed August 28, 2026. Benchmark score is a project self-report, not independently reproduced.

[129] Xingyao Wang et al., "OpenHands: An Open Platform for AI Software Developers as Generalist Agents," *ICLR 2025*, arXiv:2407.16741.

[130] Anthropic, "How the Agent Loop Works," *Claude Agent SDK documentation*, accessed August 28, 2026. Vendor documentation; cited as product fact.

[131] Thibaud Gloaguen, Niels Mündler, Mark Müller, Veselin Raychev, and Martin Vechev, "Evaluating AGENTS.md: Are Repository-Level Context Files Helpful for Coding Agents?," arXiv:2602.11988, February 12, 2026, revised June 23, 2026. Preprint.

[132] Jai Lal Lulla, Seyedmoein Mohsenimofidi, Matthias Galster, Jie M. Zhang, Sebastian Baltes, and Christoph Treude, "On the Impact of AGENTS.md Files on the Efficiency of AI Coding Agents," ICSE JAWs 2026. Task success rate explicitly not evaluated.

[133] Ali Arabat and Mohammed Sayagh, "Toward Instructions-as-Code: Understanding the Impact of Instruction Files on Agentic Pull Requests," arXiv:2606.13449. 15,549 agentic pull requests across 148 projects. Preprint.

[134] Worawalan Chatlatanagulchai, Hao Li, Yutaro Kashiwa, et al., "Agent READMEs: An Empirical Study of Context Files for Agentic Coding," arXiv:2511.12884, November 17, 2025. 2,303 files from 1,925 repositories. Preprint.

[135] "AGENTS.md," open format, stewarded by the Agentic AI Foundation under the Linux Foundation.

[136] Anthropic, "Effective Context Engineering for AI Agents," September 29, 2025. Vendor position; no efficacy figures published for the three named long-horizon techniques.

[137] Stefan Heule, Emily Jia, and Naman Jain, "Improving Agent with Semantic Search," *Cursor Blog*, November 6, 2025. Vendor-published, opposite architectural position to reference 136; each publishes evidence favoring its own approach.

[138] Kelly Hong, Anton Troynikov, and Jeff Huber, "Context Rot: How Increasing Input Tokens Impacts LLM Performance," Chroma Research, July 14, 2025. Publisher sells vector databases and has a commercial interest in the conclusion.

[139] Anthropic, "Subagents in the SDK," *Claude Agent SDK documentation*, accessed August 28, 2026. Mechanism documented; task-success benefit asserted rather than measured.

[140] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang, "Agentless: Demystifying LLM-based Software Engineering Agents," *Proceedings of the ACM on Software Engineering* (FSE 2025), arXiv:2407.01489, DOI 10.1145/3715754.

[141] Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini, "CodeMonkeys: Scaling Test-Time Compute for Software Engineering," arXiv:2501.14723. Preprint.

[142] Erik Schluntz and Barry Zhang, "Building Effective AI Agents," Anthropic, December 19, 2024. Vendor-published.

[143] Anthropic, "How We Built Our Multi-Agent Research System," June 13, 2025. Vendor-published; the same post reports ~15× token usage and states that most coding tasks involve fewer truly parallelizable subtasks than research.

[144] Walden Yan, "Don't Build Multi-Agents," Cognition, June 12, 2025. Vendor-published; the author has since publicly softened the position.

[145] Harrison Chase, "How and When to Build Multi-Agent Systems," LangChain, June 16, 2025. Vendor-published.

[146] Mert Cemri, Melissa Z. Pan, Shuyi Yang, et al., "Why Do Multi-Agent LLM Systems Fail?," *NeurIPS 2025*, arXiv:2503.13657. 1,600+ annotated traces, seven frameworks, Cohen's κ = 0.88. The one large peer-reviewed non-vendor result in this debate.

[147] Dat Tran and Douwe Kiela, "Single-Agent LLMs Outperform Multi-Agent Systems on Multi-Hop Reasoning Under Equal Thinking Token Budgets," arXiv:2604.02460, April 2, 2026. Multi-hop question answering, not software engineering. Preprint.

[148] Jiawei Xu, Arief Koesdwiady, Sisong Bei, et al., "Rethinking the Value of Multi-Agent Workflow: A Strong Single Agent Baseline," arXiv:2601.12307, January 18, 2026. Preprint.

[149] Thomas Kwa, Ben West, Joel Becker, et al., "Measuring AI Ability to Complete Long Software Tasks," METR, arXiv:2503.14499, March 19, 2025; and METR, "Task-Completion Time Horizons of Frontier AI Models," updated May 8, 2026. Measures task difficulty in human time, not autonomous run duration; measurements above 16 hours are unreliable with the current task suite.

[150] GitHub, "About GitHub Copilot Cloud Agent," *GitHub Docs*, accessed August 28, 2026. Vendor documentation; cited as product fact.

[151] OpenAI, “Run Long Horizon Tasks with Codex,” *OpenAI Developers Blog*, accessed August 28, 2026. A single documented run, not a measurement.

[152] Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping, “The Illusion of Diminishing Returns: Measuring Long Horizon Execution in LLMs,” *ICLR 2026*, arXiv:2509.09677.

[153] Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville, “LLMs Get Lost In Multi-Turn Conversation,” arXiv:2505.06120, May 9, 2025. Over 200,000 simulated conversations; tests conversational generation rather than agentic coding trajectories — the transfer is plausible and unproven.

[154] Longju Bai, Zhemin Huang, Xingyao Wang, Jiao Sun, Rada Mihalcea, Erik Brynjolfsson, Alex Pentland, and Jiaxin Pei, “How Do AI Agents Spend Your Money? Analyzing and Predicting Token Consumption in Agentic Coding Tasks,” arXiv:2604.22750. Preprint.

[155] Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan, “AI Agents That Matter,” arXiv:2407.01502, July 1, 2024. Preprint.

[156] GitHub, “spec-kit: Toolkit to Help You Get Started with Spec-Driven Development,” GitHub repository, MIT licensed. Presents no empirical evidence, benchmark, or user study; its own goals are labeled experimental.

[157] Noble Saji Mathews and Meiyappan Nagappan, “Test-Driven Development and LLM-based Code Generation,” *ASE 2024*, arXiv:2402.13521.

[158] Oorja Majgaonkar, Zhiwei Fei, Xiang Li, Federica Sarro, and He Ye, “Understanding Code Agent Behaviour: An Empirical Study of Success and Failure Trajectories,” arXiv:2511.00197, October 31, 2025. Preprint.

[159] Vantage, “The Hidden Cost Driver in Agentic Coding Sessions,” April 15, 2026. Vendor modeling, not measurement.

[160] HAL leaderboard, Princeton. Late-2024 model-scaffold pairs; cost and accuracy are not correlated across entries.

[161] Chengpeng Li, Farnaz Behrang, August Shi, and Peng Liu, “FlakyGuard: Automatically Fixing Flaky Tests at Industry Scale,” *ASE 2025*, arXiv:2511.14002. Peer-reviewed; deployed autonomously at Uber over six months. The 17.7% end-to-end figure is derived from the paper’s three reported conditional rates, not stated by the authors; plan against it rather than against the 51.8% acceptance rate.

[162] John Micco, “Flaky Tests at Google and How We Mitigate Them,” *Google Testing Blog*, May 27, 2016.

[163] Snyk, “Benchmarking Secure-and-Functional Remediation,” August 18, 2026. Vendor-published; ~150 samples, three languages, single-file scope, two runs — the vendor states the sample is too small for significance on narrow gaps.

[164] Benjamin Steenhoek, Siva Sivaraman, Renata Saldivar Gonzalez, Yevhen Mohylevskyy, Roshanak Zilouchian Moghaddam, and Wei Le, “Closing the Gap: A User Study on the Real-World Usefulness of AI-powered Vulnerability Detection & Repair in the IDE,” arXiv:2412.14306. 17 professional developers, 24 projects, >1.7M lines; conducted with Microsoft.

[165] Charles Covey-Brandt, “Accelerating Large-Scale Test Migration with LLMs,” *Airbnb Tech Blog*, March 13, 2025. First-party; no dollar cost disclosed, and long-tail files required 50–100 retry attempts.

[166] Celal Ziftci, Stoyan Nikolov, Anna Sjövall, Bo Kim, Daniele Codecasa, and Max Kim, “Migrating Code At Scale With LLMs At Google,” arXiv:2504.09691, DOI 10.1145/3696630.3728542. Twelve-month study; three developers, 39 migrations, 595 changes, 93,574 edits.

[167] Hyrum Wright, “Large-Scale Changes,” chapter 22 in *Software Engineering at Google* (Sebastopol, CA: O’Reilly, 2020). Predates AI by a decade; its shard-and-land architecture has not been re-validated for agentic work.

[168] Uber, “Uber’s Strategy to Upgrading 2M+ Spark Jobs,” *Uber Blog*, September 25, 2025. First-party. Deterministic AST transformation; the account contains no language-model use anywhere, though it does not state a rejection of the option. The “85% job migration” figure is a section heading rather than a measured automation rate.

[169] Nadia Alshahwan, Jubin Chheda, Anastasia Finogenova, Beliz Gokkaya, Mark Harman, Inna Harper, Alexandru Marginean, Shubho Sengupta, and Eddy Wang, "Automated Unit Test Improvement Using Large Language Models at Meta" (TestGen-LLM), FSE Companion 2024, arXiv:2402.09171.

[170] Christopher Foster, Rahul Gulati, Mark Harman, Inna Harper, Ke Mao, Jillian Ritchey, Hervé Robert, and Shubho Sengupta, "Mutation-Guided LLM-based Test Generation at Meta" (ACH), FSE Companion '25, arXiv:2501.12862. The 34-point acceptance spread between two teams on the same tool is the finding worth carrying.

[171] Mohammed Latif Siddiq, Zhao, Lopes, Casey, and Santos, "Security in the Age of AI Teammates: An Empirical Study of Agentic Pull Requests on GitHub," *Information and Software Technology*, arXiv:2601.00477.

[172] Rahul Gopu and David Apirian, "60 Million Copilot Code Reviews and Counting," *The GitHub Blog*, March 5, 2026. Vendor-published; comment acceptance rate, merge-time effect, and code-quality delta are all absent.

[173] Danny Hsu, M. Neu, M. Farrag, and R. Kindi, "Leveraging AI for Efficient Incident Response," *Engineering at Meta*, June 24, 2024. 42% is ranking accuracy on a constrained candidate set, not autonomous resolution; no MTTR delta published.

[174] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan, "Recommending Root-Cause and Mitigation Steps for Cloud Incidents Using Large Language Models," *ICSE 2023*, arXiv:2301.03797.

[175] Zhao, Tang, and Qian, "Do Deployment Constraints Make LLMs Hallucinate Citations?," arXiv:2603.07287, March 7, 2026. 17,443 generated citations; cited here as the closest quantified analogue to documentation risk. Preprint.

[176] Danilo Poccia, "Amazon Q Code Transformation," *AWS News Blog*, November 28, 2023, updated April 30, 2024. The widely cited "30,000 applications / 4,500 developer-years / $260M" figure originates in executive statements of August 2024 with no published methodology and should be cited as an executive claim, never as a measurement.

[177] Goran Petrović and Marko Ivanković, "State of Mutation Testing at Google," *ICSE-SEIP '18*.

[178] Goran Petrović, Marko Ivanković, Gordon Fraser, and René Just, "Practical Mutation Testing at Scale: A View from Google," *IEEE Transactions on Software Engineering*, August 2021, DOI 10.1109/TSE.2021.3107634.

[179] Zhao, Zhou, and Cohen, "Do Coverage and Mutation Scores of LLM-Generated Test Suites Correlate with Their Effectiveness? (Replicability Study)," *PACMSE*, July 2026, DOI 10.1145/3832093.

[180] Farima Mehrpour and Thomas D. LaToza, "Can Static Analysis Tools Find More Defects? A Qualitative Study of Design Rule Violations Found by Code Review," *Empirical Software Engineering* 28, no. 1 (November 2022), DOI 10.1007/s10664-022-10232-4.

[181] Alberto Bacchelli and Christian Bird, "Expectations, Outcomes, and Challenges of Modern Code Review," *ICSE 2013*. Still the best evidence on what review actually does versus what practitioners believe it does.

[182] Wang, Pradel, and Liu, "Are 'Solved Issues' in SWE-bench Really Solved Correctly? An Empirical Study," arXiv:2503.15223, March 19, 2025. Preprint.

[183] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang, "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation" (EvalPlus), *NeurIPS 2023*, arXiv:2305.01210.

[184] Jesse Toth, "Scientist," *GitHub Engineering Blog*, February 3, 2016, updated December 3, 2020. The canonical documented shadow-verification implementation; no defect-catch figures published.

[185] Gabor, Lynch, and Rosenfeld, "EvilGenie: A Reward Hacking Benchmark," arXiv:2511.21654v2, May 17, 2026. Preprint; Cambridge Boston Alignment Initiative and MIT FutureTech.

[186] Salim, Latendresse, Khatoonabadi, and Shihab, "Tokenomics: Quantifying Where Tokens Are Used in Agentic Software Engineering," arXiv:2601.14470, January 20, 2026. n = 30 tasks, one framework, one model; its "review" stage is a model reviewing model output, not CI compute. Preprint.

[187] Atif Memon, Zebao Gao, Bao Nguyen, Sanjeev Dhanda, Eric Nickell, Rob Siemborski, and John Micco, "Taming Google-Scale Continuous Testing," *ICSE-SEIP 2017*.

[188] Haider and Zimmermann, "Understanding Dominant Themes in Reviewing Agentic AI-authored Code," *MSR '26*, arXiv:2601.19287. 19,450 inline review comments across 3,177 agent-authored pull requests.

[189] He, Shao, Chen, Gao, Zhang, and Sheng, "Use Property-Based Testing to Bridge LLM Code Generation and Validation," arXiv:2506.18315. The paper assumes rather than demonstrates that model-generated properties are more reliable than model-generated implementations. Preprint.

[190] Ma, Zhang, Cao, Liu, Zhang, Luo, Zhang, and Chen, "Rethinking Verification for LLM Code Generation: From Generation to Testing," arXiv:2507.06920v2, July 2025. Names the "homogenization trap." Preprint.

[191] Kubernetes SIG Apps, "Agent Sandbox," kubernetes-sigs/agent-sandbox, v0.4.6, May 14, 2026. Pre-1.0.

[192] Anthropic, "Claude Code Sandboxing," Engineering blog, October 20, 2025, and "Choose a Sandbox Environment," Claude Code documentation. Vendor-published; the sandbox runtime is open-sourced and the architecture claims are verifiable in the released code.

[193] Matt Mathew, Prasad Borole, Meng Huang, Sergey Burykin, Gaurav Goel, and Bayard Walsh, "Solving the Identity Crisis for AI Agents," *Uber Engineering Blog*, May 21, 2026. First-party; the strongest published enterprise agent-identity implementation.

[194] Amit Arora and Omri Shiv, "Governing AI Assets at Scale with MCP Gateway and Registry," *AWS Open Source Blog*, June 17, 2026. Vendor-published, Apache-2.0, genuinely implementable.

[195] Spotify, "Portal MCP / Actions Registry," Backstage documentation; and Tyson Singer, "Introducing Xirp," Spotify Portal blog, August 10, 2026. Vendor-published; internal adoption figures self-reported.

[196] Natalie Lung, "Uber Caps Usage of AI Tools Like Claude Code to Manage Costs," *Bloomberg*, June 2, 2026 (paywalled), corroborated by Simon Willison, June 3, 2026.

[197] CNCF and SlashData, "Platform Engineering Tools Maturing as Organizations Prepare for AI-Driven Infrastructure," March 24, 2026. Survey of 400+ professional developers, fielded Q4 2025.

[198] Gartner, "2026 Hype Cycle for Agentic AI," April 15, 2026. 17% of organizations have deployed AI agents; over 60% expect to within two years. Analyst-published; underlying sample not disclosed publicly.

[199] Yasmin Moslem and John D. Kelleher, "Dynamic Model Routing and Cascading for Efficient LLM Inference: A Survey," arXiv:2603.04445v2, April 21, 2026. Reported figures are benchmark results from individual papers, not enterprise production results. Preprint.

[200] GitHub, "About Premium Requests," *GitHub Docs*. Vendor documentation; cited as product fact for the quota mechanism.

[201] Northflank, "AI Sandbox Pricing Comparison (2026)," May 5, 2026. Vendor comparing itself against competitors; cite only the third-party list prices, which are independently checkable.

[202] Sergio De Simone, "AI Code Review at Scale: LinkedIn's Multi-Agent Approach," *InfoQ*, August 22, 2026. 5,230 sampled review comments across 1,727 pull requests.

[203] Bowen Qin and Yi Xie, "Agent Retrieval Bench: Evaluating Repository Context Retrieval for Coding Agents," arXiv:2607.24882, July 2026. 427 samples across 25 repositories. Preprint.

[204] Taj Shorter, "Inside Shopify's AI-First Engineering Playbook," Bessemer Venture Partners, April 1, 2026. Third-party interview; figures self-reported by Shopify with no methodology.

[205] Microsoft, "What Is Microsoft Entra Agent ID," Microsoft Learn, documentation dated April 14, 2026, updated June 24, 2026. No formal general-availability date is stated in the documentation; do not assert GA.

[206] modelcontextprotocol/registry, community MCP registry, in preview with the API frozen at v0.1.

[207] Erik Brynjolfsson, Bharat Chandar, and Ruyu Chen, "Canaries in the Coal Mine? Six Facts about the Recent Employment Effects of Artificial Intelligence," Stanford Digital Economy Lab, August 2026; and "Canaries, Interest Rates, and Timing," February 9, 2026. ADP administrative payroll microdata, balanced panel of 3.5–5 million employees per month — the highest-quality evidence in this domain.

[208] SignalFire, *State of Tech Talent Report 2026*. Assigns primary causation to the end of the zero-interest-rate era rather than to AI; genuinely disagrees with reference 207 on cause while agreeing on fact.

[209] Hao-Ping Lee, Advait Sarkar, Lev Tankelevitch, Ian Drosos, Sean Rintel, Richard Banks, and Nicholas Wilson, "The Impact of Generative AI on Critical Thinking," *CHI 2025*, DOI 10.1145/3706598.3713778. n = 319 knowledge workers, 936 usage instances.

[210] Raja Parasuraman and Dietrich H. Manzey, "Complacency and Bias in Human Use of Automation: An Attentional Integration," *Human Factors* 52, no. 3 (June 2010): 381–410, DOI 10.1177/0018720810376055.

[211] Atlassian, *State of Developer Experience 2025*, July 9, 2025. n = 3,500 developers and managers across six countries. Vendor-published.

[212] Matthew Skelton, "Team Topologies as the Infrastructure for Agency with AI," QCon London, March 2026, reported *InfoQ*, March 2026; and Olivier Wulveryck, "Who Does What? Team Topologies for the Agentic Platform," June 22, 2026. Conference-stage and practitioner reasoning respectively; neither is measured.

[213] Duolingo, AI-first memo (April 2025) and subsequent clarifications, with the substantive reversal reported in *Fortune*, April 13, 2026. Primary memo text not retrievable; all memo language is secondhand through contemporaneous reporting.

[214] Klarna, "Klarna AI Assistant Handles Two-Thirds of Customer Service Chats in Its First Month," press release, February 27, 2024; reversal reported *Entrepreneur*, May 2025; steady state reported *Fortune*, October 10, 2025. The agent-equivalence figure is 700 in the primary source and 800 in later coverage; use 700.

[215] Salesforce, statements by Marc Benioff on the Q4 FY25 earnings call, February 26, 2025; hiring announcement reported *Fortune*, April 27, 2026; and earnings call, May 28, 2026. The 30% engineering productivity claim has never been substantiated or repeated with a methodology.

[216] Nicholas Carlini, "Building a C Compiler with a Team of Parallel Claudes," Anthropic Engineering, February 5, 2026. n = 1 human; vendor-published.

[217] Microsoft, *Work Trend Index 2025: The Year the Frontier Firm Is Born*, April 24, 2025. Coined "human-agent ratio" and explicitly supplied no formula, benchmark, or tested figure.

[218] Ipek Ozkaya, Anita Carleton, Sebastián Echeverría, et al., *The AI Adoption Maturity Model v1.0*, Carnegie Mellon Software Engineering Institute, June 30, 2026. General AI adoption rather than agentic engineering specifically.

[219] Sabry E. Farrag, "The Productivity-Reliability Paradox: Specification-Driven Governance for AI-Augmented Software Development," University of East London, arXiv:2605.01160, May 2026. Multivocal review of 67 sources; a proposal, not validated in the field. Preprint.

[220] Indeed Hiring Lab software development postings index, FRED series IHLIDXUSTPSOFTDEVE, 74.57 as of August 21, 2026; and Lightcast forward-deployed-engineer posting data reported in *IT Brew*, December 19, 2025.

[221] IBM entry-level hiring statements, reported *Axios*, February 13, 2026. Executive claim; no published outcome data.

[222] Coinbase, statements by Brian Armstrong on the *Cheeky Pint* podcast, August 22, 2025, reported *TechCrunch*, August 22, 2025. The number of engineers dismissed was never disclosed; the reported AI-authored code percentage is unverified.

[223] Tobi Lütke, internal memo posted publicly to X, April 7, 2025, reported *Inc.*, April 2025. No reversal reported as of August 2026, and no published outcome data.

[224] Nx Team, security advisory GHSA-cxm3-wv7p-598c, nrwl/nx, August 27, 2025. First-party; contains the incident timeline and the enumerated remediations.

[225] Snyk, "Weaponizing AI Coding Agents for Malware in the Nx Malicious Package," August 2025. Vendor-published; reproduces the agent-invocation code verbatim.

[226] StepSecurity, "Supply Chain Security Alert: Popular Nx Build System Package Compromised with Data-Stealing Malware," August 27, 2025. Vendor-published; per-version publish timeline in UTC.

[227] Wiz Research, "s1ngularity Supply Chain Attack," August 27, 2025, and "s1ngularity's Aftermath," September 2025. Vendor-published telemetry analysis; the refusal-rate findings are uncorroborated, and the firm revised its own second-phase repository count upward mid-investigation.

[228] Ashish Kurmi (StepSecurity) and Charlie Eriksen (Aikido), quoted in Connor Jones, "Supply Chain Attack Hits Nx Build System," *The Register*, August 27, 2025.

[229] Cybersecurity and Infrastructure Security Agency, "Supply Chain Compromises Impact Nx Console and GitHub Repositories," alert, May 28, 2026. A separate, later incident affecting the same ecosystem; CVE-2026-48027.

[230] Jason Lemkin, public posts on X and SaaStr, July 2025, including the retrospective. Single-participant self-published account; figures are as reported and were not independently verified.

[231] Amjad Masad, public thread on X, July 21, 2025, with remediation detail corroborated in Connor Jones, "Replit Responds," *The Register*, July 22, 2025, and Fast Company, July 21, 2025. No written postmortem was subsequently published.

[232] Andrew Cornwall (Forrester Research), Matthew Flug (IDC), and Torsten Volk (Enterprise Strategy Group), quoted in Beth Pariseau, "Replit AI Agent Snafu 'Shot Across the Bow' for Vibe Coding," *TechTarget*, July 2025.

[233] AI Incident Database, incident 1152. Characterizes the events as reported and alleged; the claims have not been independently verified.

[234] Uber, "How Uber Executed a JUnit Migration at Massive Scale," *Uber Blog*, April 7, 2026. First-party; contains the statement that generative AI was attempted for the migration and abandoned in favor of deterministic transformation.

[235] Spotify, "Fleet Management at Spotify (Part 3): Fleet-wide Refactoring," *Spotify Engineering*, May 2023; and "Part 1: Spotify's Shift to a Fleet-First Mindset," April 2023. First-party; no language models are described in either, and no failure or rollback rate is published.

[236] Spotify, "Background Coding Agents: Context Engineering," *Spotify Engineering*, November 2025. First-party.

[237] Spotify, "Background Coding Agents: Predictable Results Through Strong Feedback Loops," *Spotify Engineering*, December 2025. First-party; source of the judge-veto rate.

[238] Devon Edwards Joseph, "Background Coding Agents: Supercharging Downstream Consumer Dataset Migrations," *Spotify Engineering*, April 22, 2026. First-party; the manual-effort comparison is an estimate.

[239] Nadia Alshahwan, Mark Harman, Inna Harper, Alexandru Marginean, Shubho Sengupta, and Eddy Wang, "Assured LLM-Based Software Engineering," arXiv:2402.04380, February 6, 2024. The methodological parent of the two Meta deployment papers.

[240] Mark Harman, Peter O'Hearn, and Shubho Sengupta, "Harden and Catch for Just-in-Time Assured LLM-Based Software Testing: Open Research Challenges," FSE '25 Companion, arXiv:2504.16472. Source of the statement that only four of the six claimed assurances are verifiable guarantees.

[241] Itamar Friedman, “We Created the First Open-Source Implementation of Meta’s TestGen-LLM,” Qodo (formerly CodiumAI), May 20, 2024. Vendor-published by a competitor; the only substantive independent engagement with the TestGen-LLM paper located.

[242] Uber, “Running a Software Factory Efficiently at Uber Scale,” *Uber Blog*, August 27, 2026. First-party; adoption and cost-trend figures self-reported.

[243] Reddit thread by u/NegativeWeb1, May 2025, and coverage in Habr, “On Reddit, They Discovered That AI Copilot on GitHub Is Slowly Driving Microsoft Employees Crazy,” May 21, 2025; primary artifact (Copilot-authored, opened May 20, 2025, closed unmerged). Practitioner reaction, not an evaluation.

Version 1.1, September 2026. This framework is offered as practitioner guidance and does not constitute legal, regulatory, or financial advice. Standards versions, regulatory dates, and published figures were verified against primary sources in September 2026 and move quickly; verify all regulatory mappings against current obligations in your jurisdiction before relying on them. Where a source is vendor-published, correlational, self-reported, or an unreviewed preprint, the reference entry says so. Appendix C lists the pending events that would occasion a revision.

