REPORTS LAB · REPORT №02 · A READINESS FIELD GUIDE

The Agentic *Enterprise*

A long-form, evidence-based guide to agentic AI in the enterprise — what changes when software stops answering and starts acting, what governance survives that transition, and what the first ninety days of a serious agent program actually look like.

40 CHAPTERS · THREE PARTS

INTERACTIVE SCORECARD · 4 PILLARS · 5 LEVELS

Contents

PART I

The Shape of Agentic AI

Twelve chapters establishing what agentic AI actually is, what it costs, how it fails, and the regulatory and threat perimeter you are operating inside.

CHAPTER 01

The Shift

From copilots to colleagues — what changes when software starts to act

> Something changed in the relationship between software and the people who use it. For most of computing's history, software was a remarkably patient thing: it waited. You typed, it responded; you clicked, it moved. Even the first generation of large-language-model products — the chatbots, the copilots, the completion engines — preserved this arrangement. The human was still the one deciding every step. What is different about an agent is precisely that: the model now decides the next step. That single reversal carries more operational, ethical, and economic weight than almost any other shift in enterprise technology since the move to the cloud.

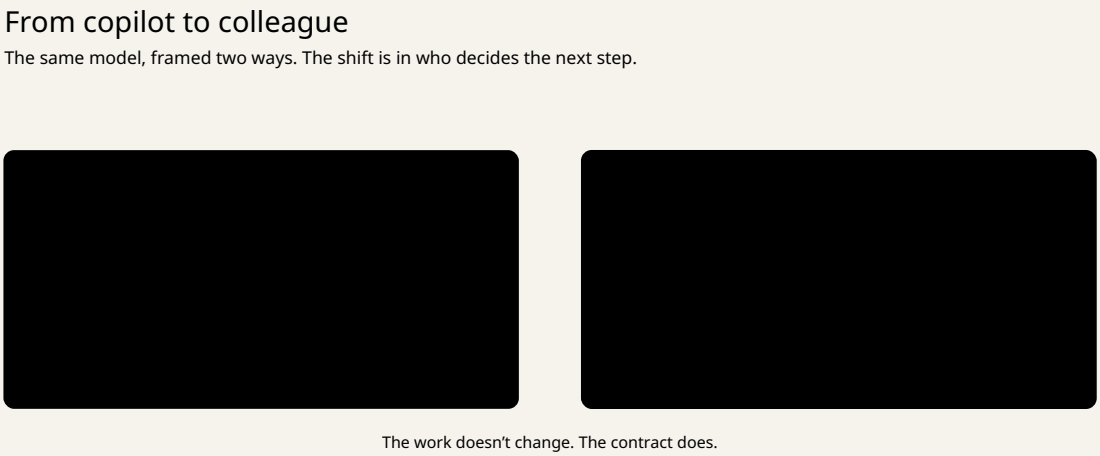

FIGURE 1.1COPILOT TO AGENT — SAME MODEL, DIFFERENT CONTRACT.

The Old Contract

The history of enterprise software is, at its core, a history of waiting. Mainframes waited for batch jobs. Client-server applications waited for keystrokes. Even web APIs — for all their promise of connectivity — waited politely for a caller. This passivity was not a bug. It was a carefully maintained guarantee: the human remained the proximate cause of every action taken in the system. When something went wrong, the trail of responsibility led, always, back to a person at a keyboard.

The large language model disrupted that contract at the linguistic layer first. Suddenly, the system could draft — it could produce text that looked authoritative, that could slip past a cursory review and land in a sent folder. But even there, the human had to press send. The

copilot was aptly named: it sat in the right seat, could handle the controls when asked, and handed them back. The pilot was still responsible.

Agents change the seating arrangement entirely. When you brief an agent on a goal — close this support ticket by finding the relevant documentation, drafting a reply, and updating the CRM record — you are not typing a prompt. You are issuing a delegation. The agent will read, search, draft, call tools, and write records. It will do all of this without asking permission at each step. You are, for the duration of that task, more manager than operator.

What Actually Changes

The practical differences between a copilot and an agent are not merely philosophical. They are architectural and operational. A copilot interaction is stateless in the meaningful sense: each prompt is self-contained, the blast radius of any single error is bounded, and the human review step is the natural firewall. An agent interaction is, by contrast, stateful, extended, and consequential. Actions accumulate. A mistake in step two doesn't just produce a bad paragraph — it sends the agent down a wrong path that may take dozens of tool calls to course-correct, or may never be corrected at all if no one is watching.

This is not a hypothetical concern. When Anthropic introduced computer use — the capability that allows Claude to navigate desktop applications, click buttons, and fill forms — it published explicit warnings about the irreversibility of actions in certain environments. When OpenAI introduced Operator, its web-browsing agent, it built in confirmation checkpoints for actions deemed consequential. Both companies recognized, and built around, the same principle: agency introduces irreversibility, and irreversibility demands governance.

The implications run deeper than safety. They run through accountability (who answers when the agent makes a costly mistake?), through auditability (can you reconstruct what the agent decided and why?), and through economics (what does an hour of agentic work actually cost, and what does it return?). Each of these questions has a familiar shape — enterprises have answered versions of them for automation systems before — but the specific texture of agentic AI makes them harder. The agent reasons in natural language, which is harder to inspect than a flowchart. It uses tools dynamically, which is harder to audit than a fixed API call sequence. It can be manipulated through the content it reads, a vulnerability that no previous class of software shared.

Early Systems, What We Learned

The first production agentic systems appeared not in enterprise IT but in software engineering. Devin, released by Cognition AI in early 2024, could take a GitHub issue and close it — reading code, writing tests, running the test suite, committing changes. OpenHands (formerly OpenDevin), the open-source equivalent, followed weeks later. Both demonstrated something important: agents could navigate genuinely complex, multi-step technical workflows. They also demonstrated something humbling: they failed in ways that experienced engineers found

surprising and non-obvious. They wrote code that passed tests but broke production. They made confident assumptions about environment state that turned out to be wrong. They sometimes looped, burning API credits and time on circular reasoning.

Claude Code, Anthropic's terminal-native coding agent, and GitHub Copilot Workspace took a more conservative approach: tighter scope, explicit human checkpoints, lower autonomy budgets. The lesson from the first wave was that autonomy is not binary, and that matching the autonomy level to the task's risk profile is more valuable than maximizing autonomy for its own sake. By 2025, enterprise platforms had absorbed this lesson. Salesforce Agentforce and ServiceNow AI Agent Fabric both ship with explicit autonomy controls, audit logging, and human-override mechanisms — engineering choices that reflect operational hard lessons, not marketing preferences.

The Enterprise Stakes

Why does any of this matter to a large organization? Because the value proposition of agentic AI — the thing that makes it genuinely transformative rather than another productivity increment — is precisely its ability to operate at scale, continuously, across systems, without constant human shepherding. A copilot that helps a customer service agent write better replies is valuable, but its value is proportional to the number of agents who use it and the hours they spend with it. An agent that handles the routine tier-one support queue end-to-end — reading tickets, looking up account history, drafting resolutions, escalating edge cases — is valuable proportional to the volume of work, which may be orders of magnitude larger.

That is also why the stakes are higher. Scale amplifies errors as readily as it amplifies value. An agent with a misconfigured policy, or a prompt-injection vulnerability in the documents it reads, or an overly permissive set of tool access rights, does not make one mistake. It makes that mistake ten thousand times before anyone notices.

The organizations that will extract durable value from agentic AI are not necessarily those with the most advanced models. They are those with the governance structures, the integration plumbing, the evaluation frameworks, and the cultural readiness to deploy agents at scale with confidence. That is what this report is about.

> *"The model is the least interesting part of the problem. What is hard — what has always been hard — is the surrounding system: the data, the integrations, the policies, the oversight. Agents make that problem bigger and more urgent, not smaller."*

A Map of What Follows

Part I of this report builds the conceptual and threat-model foundation. It answers what an agent is, how autonomy should be classified, what the economics look like, how it fails, and what legal and regulatory perimeter it operates inside. Part II turns to the four pillars of readiness:

governance, orchestration, use case identification, and integration. Part III offers the operational instruments — the scorecard, the roadmap, the operating model — that turn understanding into action.

The audience is a CIO, a CISO, a chief data officer, or a senior technology leader who has moved past asking what is this? and arrived at how do we do this responsibly, at scale, in a way we can defend to regulators and the board? The answer is not simple. But it is knowable. And the enterprises that know it first will hold an advantage that compounds.

> OPERATOR'S NOTE
>
> When briefing a board or executive committee on agentic AI, resist the temptation to start with the technology. Start with the contract change: the organization is moving from software that waits to software that acts. That single sentence captures the governance challenge better than any technical diagram, and it prompts the right questions about accountability, audit, and risk appetite.

CHAPTER 02

Anatomy of an Agent

Plan, act, observe, remember — the four moving parts

> Beneath the surface of every agent — whether it books travel, reviews contracts, or writes production code — the same four-part machinery is running. The agent plans, acts, observes the results of its actions, and remembers what it has learned. Then it plans again. This loop, deceptively simple in description, is where virtually all of the interesting behavior emerges, and where virtually all of the failures begin. Understanding the mechanics of each component — not at the level of a research paper but at the level of an engineering and governance decision — is the prerequisite for everything else in this report.

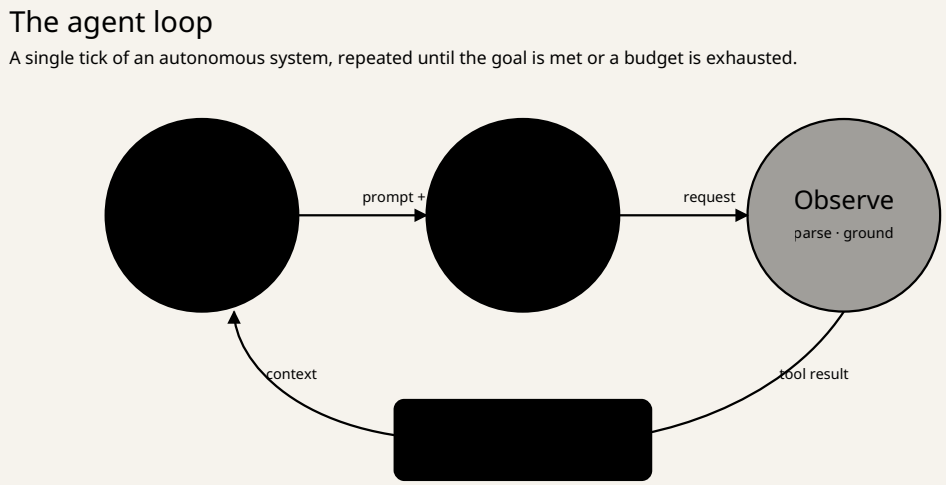

FIGURE 2.1THE AGENT LOOP. PLAN, ACT, OBSERVE, REMEMBER — AND THEN PLAN AGAIN, HAVING LEARNED SOMETHING. MOST AGENTIC FAILURES LIVE ON THE ARROWS, NOT IN THE CIRCLES.

The Planning Layer

Planning is the step where a model transforms a goal into a sequence of actions. In the simplest case, this is a single inference call that produces a short to-do list. In more sophisticated systems, planning is itself a multi-step process: a high-level planner decomposes the goal into sub-goals, each of which is handed to a more specialized executor. The ReAct paradigm — Reasoning and Acting interleaved in a single prompt — was an early demonstration that models could plan and act in the same context window. More recent architectures, like those underlying Anthropic's Claude Code and OpenAI's Codex-based tools, use explicit scratchpad reasoning that is stripped before the final output is produced.

The planning layer is where model quality matters most. A model that reasons well — that can decompose ambiguous goals, identify dependencies between steps, recognize when a plan needs revision — will complete tasks that a weaker model fails at, even with identical tool access. But planning quality is also where the first class of failures appears: models that confidently construct plans based on incorrect assumptions about the environment, or that over-plan (generating elaborate multi-step sequences for tasks that could be accomplished in one tool call), or that under-plan (diving into execution without adequate decomposition, creating situations that are difficult to recover from).

Enterprise architects need to understand the planning layer not only to evaluate model capability but to design appropriate oversight. A plan that is legible — that can be inspected before execution begins — enables a class of human-in-the-loop governance that is impossible when planning and execution are fused. Some orchestration frameworks, including LangGraph and CrewAI, expose explicit plan objects for this reason. Others, following the React paradigm, interleave planning and execution in a way that makes pre-execution review impractical. Neither approach is universally correct; the choice depends on the reversibility of the actions involved.

The Act Layer

Acting is the step where the agent reaches outside its own context window and changes something in the world. This is the step that distinguishes an agent from a chatbot, and it is the step that creates real risk. Tool calls are the primary mechanism: the model generates a structured call to a function, API, browser, code interpreter, or external service, and the result is returned to the context.

The range of tools available to modern agents has expanded dramatically. Early tool-augmented models could search the web and run simple computations. Current systems — using the Model Context Protocol (MCP), which has become the de facto standard for agent-tool integration — can interact with essentially any system that can expose an API: CRMs, ERPs, file systems, databases, code repositories, email, calendars, and increasingly, physical-world interfaces like browser-controlled desktops and robotic actuators.

The governance implications of tool access are proportional to the power of the tools. An agent that can only read documents has a very different risk profile from one that can write to a database, send emails on behalf of a user, or execute code in a production environment. The principle of least privilege — grant only the minimum tool access necessary for the task — is as important in agentic system design as it is in conventional access control. In practice, it is harder to enforce: tool permissions are often configured once at deployment time and not revisited as the agent's operational scope evolves.

The Observe Layer

Observation is the step where the agent processes the results of its actions. A tool call returns a response — a database query result, an API response body, a web page's text content, a code

execution output — and the agent must parse that response, determine whether its action succeeded, and update its understanding of the world accordingly.

This step sounds passive but is anything but. The observe layer is where a significant class of vulnerabilities lives. The agent is, by design, trusting the content it receives from tool calls. If that content has been manipulated — if a document the agent retrieves from the web contains instructions designed to redirect its behavior, if a database record has been poisoned with adversarial text — the agent may incorporate those instructions into its planning and produce behavior that its operators neither intended nor anticipated. This is the mechanism behind indirect prompt injection, one of the most operationally significant security risks in agentic systems and a central concern of the OWASP LLM Top 10.

Designing robust observation means building not just the capability to receive tool results but the capability to verify them. Does the result make sense given the context? Is it plausible that this external source would return this content? Should the agent flag this result for human review before acting on it? These are design questions that current frameworks leave largely to the application developer, and they are questions that few developers ask with sufficient rigor before deployment.

The Memory Layer

Memory is what separates a stateful agent from a stateless one, and a useful agent from an irritating one. The simplest form of agent memory is the context window itself: everything the model has seen in the current interaction is, in principle, available to it. But context windows are finite (even very large ones impose latency and cost penalties), and they are ephemeral — they do not persist across sessions. For agents that operate over extended time horizons, on tasks that span hours or days, or in environments where they need to recall decisions made in prior sessions, external memory stores are required.

Memory architectures typically distinguish at least three layers. **Short-term memory** — the current context window — holds the immediate task state. **Long-term memory** — typically a vector database or a structured key-value store — holds persistent facts, prior decisions, user preferences, and domain knowledge that should survive across sessions. **Episodic memory** — a log of what the agent actually did and what the results were — is the most underbuilt of the three in current systems, yet it is the most important for both learning and audit.

From a governance perspective, memory is where data residency, retention, and privacy obligations land. An agent that accumulates a long-term memory of a user's decisions, preferences, and conversations is, in effect, building a profile. That profile is subject to the same GDPR and CCPA obligations as any other personal data store — but the tooling for managing agent memory with the same rigor as a conventional database is still immature. Organizations deploying agents with persistent memory should, at minimum, define explicit retention policies, classification labels for stored memories, and audit mechanisms before they discover these gaps through a regulator rather than an internal review.

> *"Most agentic failures don't live in the circles — in the plan, the act, or the observe step taken individually. They live on the arrows: in the translation between observation and the next plan, where wrong assumptions compound, and in the memory layer, where bad facts get entrenched."*

How the Loop Runs

The four components do not execute once. They execute in a loop — plan, act, observe, update memory, plan again — until the agent determines that the goal has been met, that the goal cannot be met, or that it has reached its resource budget (in tokens, tool calls, or elapsed time). The loop is the source of both the power and the risk.

Power: the loop allows an agent to adapt. If a tool call fails, the agent can try a different approach. If new information changes the picture, the agent can revise the plan. This adaptivity is what makes agents useful for open-ended tasks where the path is not known in advance. Risk: the loop can diverge. An agent that loses track of its goal, or that develops an incorrect model of the environment, or that is manipulated through adversarial tool outputs, can run for many iterations in a direction that is costly or irreversible before any oversight mechanism catches it.

Practical agent deployments manage loop risk through a combination of resource budgets (maximum token spend, maximum tool call count, maximum elapsed time), explicit goal-completion checks, and human-in-the-loop confirmation gates for actions above a configurable risk threshold. The specific values of these parameters are among the most consequential engineering decisions in agentic system design — and among the least discussed in vendor documentation.

> ARCHITECTURE NOTE
>
> When specifying an agent system for procurement or internal development, require explicit documentation of the four components: the planning mechanism and its legibility, the tool catalog and permission model, the observation validation approach, and the memory architecture with its retention and classification policy. A vendor who cannot answer these questions clearly has likely not thought through the governance implications of their system.

CHAPTER 03

The Spectrum of Autonomy

Five rungs from automation to autonomy, and where most enterprises actually live

> Not all agents are equally autonomous. This observation sounds obvious, but its implications are underappreciated in most enterprise AI discussions, where "agent" is used loosely to describe everything from a macro that runs when you click a button to a system that autonomously negotiates contracts on behalf of the organization. The difference is not cosmetic. It determines the governance overhead, the integration requirements, the appropriate risk controls, and the realistic timeline to production. Before deploying anything that calls itself an agent, a well-run organization should be able to specify exactly where on the autonomy spectrum it sits — and why that level is appropriate for the task at hand.

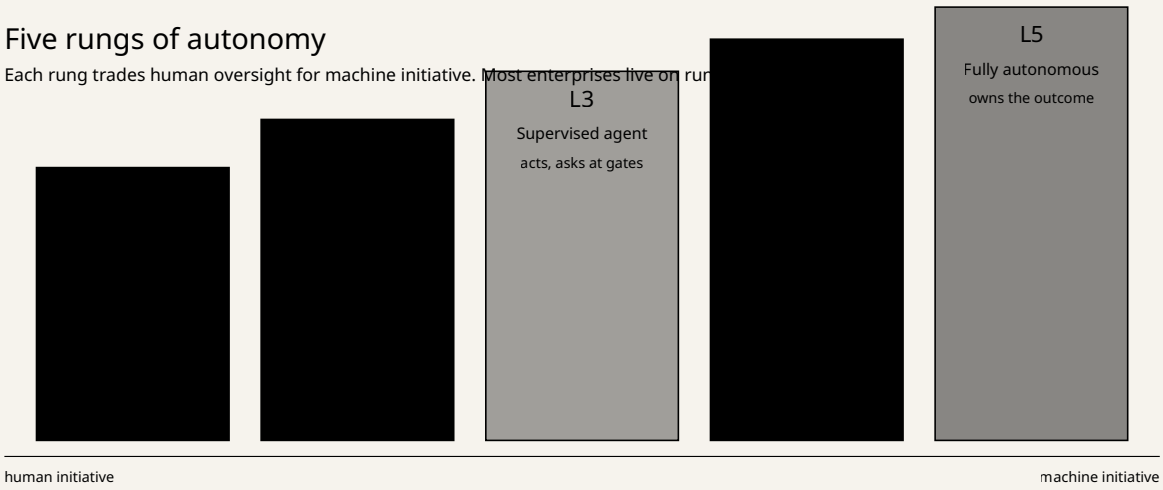

FIGURE 3.1FIVE RUNGS OF AUTONOMY. MOST ENTERPRISE PILOTS IN 2026 SIT ON RUNG 2; MOST ENTERPRISE VALUE LIVES ON RUNGS 3 AND 4.

Five Rungs

The framework presented here uses five levels, roughly analogous to the SAE autonomy levels for self-driving vehicles but adapted for enterprise AI. The analogy is imperfect — enterprise agents operate in a very different risk environment than autonomous cars — but the underlying logic is the same: each level represents a transfer of initiative and decision authority from human to machine, and that transfer has implications that run through safety, accountability, and system design.

Level 1 — the assistant is the familiar chatbot or completion engine. It replies when asked. Every step is human-initiated. The model's output is informational; the human decides what to do with

it. Risk is low; blast radius is bounded by the human review step. Most enterprise AI deployments today are still at this level, and for many use cases — exploratory research, document summarization, code completion — this is the right level.

Level 2 — the copilot adds the ability to draft actions, not just words. A copilot might propose a calendar entry, draft an email reply, or generate a code commit. The human approves before anything is executed. This is the dominant pattern in commercial enterprise AI as of 2025: Microsoft 365 Copilot, Salesforce Einstein Copilot, GitHub Copilot. The key distinction from Level 1 is that the model is now proposing actions, not just text, but the human retains final authority over execution.

Level 3 — the supervised agent executes autonomously within a task but pauses at defined checkpoints to confirm with a human before taking consequential actions. The agent might complete the first three steps of an expense report submission without asking, but pause before submitting to the CFO's approval queue. ServiceNow AI Agent Fabric and Salesforce Agentforce both operate with explicit Level 3 controls, allowing organizations to configure which actions require confirmation and which can proceed automatically.

Levels Four and Five

Level 4 — bounded autonomy is where the agent operates within a defined action budget and permission scope without routine human checkpoints. It can take consequential actions — sending emails, updating records, executing code, provisioning resources — as long as those actions fall within its pre-defined permission envelope. Human oversight is retrospective rather than prospective: the agent acts, logs its actions, and humans review logs rather than approving each step. This is the level at which agentic AI begins to deliver genuinely transformative labor economics, because it no longer requires a human time-budget proportional to the agent's work volume. It is also the level at which governance requirements become significantly more demanding.

Level 5 — full autonomy is the research frontier, not the enterprise present. A Level 5 system owns outcomes across open-ended time horizons, makes strategic as well as tactical decisions, and interacts with other agents and systems without pre-defined permission envelopes. No commercial enterprise deployment operates at Level 5 today; the few systems that approach it — long-horizon research agents, experimental multi-agent swarms — are confined to sandboxed environments with extensive monitoring. The governance frameworks needed for Level 5 deployment do not yet exist in final form.

The honest assessment of most enterprise AI programs in 2026 is that they are between Levels 2 and 3: capable of proposing actions and executing simple workflows, but not yet trusted to operate in consequential domains with Level 4 autonomy. Closing that gap — deploying Level 3 and 4 agents in production, with governance frameworks adequate to the risk — is the central challenge that the rest of this report addresses.

Matching Level to Task

The most common failure mode in enterprise agentic programs is mismatched autonomy: deploying a Level 4 system on a task that demands Level 3 oversight, or, more commonly, deploying a Level 2 system on a task where Level 4 is achievable and would deliver the economics that justified the program in the first place. Both mistakes are costly.

Over-autonomy — more initiative than the governance infrastructure can support — leads to the headlines: agents that take irreversible actions, send emails they shouldn't, or modify records in ways that take weeks to unwind. Under-autonomy — keeping a Level 2 leash on a task that could safely operate at Level 4 — produces the quieter failure of a system that delivers insufficient return to justify its cost and complexity, which eventually becomes the justification for defunding the program.

The matching process requires a structured analysis of two variables: the reversibility of the agent's actions, and the quality of the oversight feedback loop. High reversibility and tight feedback loops argue for higher autonomy. Low reversibility or sparse feedback loops argue for lower autonomy and more explicit checkpoints. This analysis should be documented, reviewed, and revisited as the agent's operational scope evolves — a task that most current governance frameworks do not yet explicitly require but that the Singapore Model AI Governance Framework for Agentic AI, published in January 2026, addresses directly.

The Governance Multiplier

Each rung on the autonomy ladder multiplies the governance overhead. A Level 1 system needs basic content safety and data handling policies. A Level 2 system adds action proposal logging. A Level 3 system requires explicit checkpoint logic, human escalation paths, and audit trails for the decisions made at each gate. A Level 4 system requires all of the above plus runtime behavioral monitoring, anomaly detection, kill switches, and retrospective audit capable of reconstructing the full decision chain for any given action.

This governance multiplier is not a reason to stay at Level 2 forever. It is a reason to build governance infrastructure in parallel with autonomy capability, rather than retrofitting governance onto an agent that has already been deployed in production. The organizations that have done this well — that have shipped Level 4 agents into consequential workflows — have universally built their governance infrastructure before, or at least alongside, their autonomy capability. The organizations that have done this badly have discovered the governance gap through an incident rather than through planning.

> *"The autonomy level is not a model property. It is a system property, determined by the permission envelope, the checkpoint logic, the monitoring infrastructure, and the kill switch — not by the model's capability in isolation."*

CLASSIFICATION GUIDE

When inventorying existing or planned agent deployments, assign each an explicit autonomy level using the five-rung framework and document the assignment. This simple exercise routinely surfaces mismatches — systems labeled "agent" that are operating at Level 1, and systems expected to operate at Level 2 that have been architected with no checkpoints and are effectively running at Level 4. The inventory is also the foundation of any regulator-facing governance documentation.

CHAPTER 04

From Models to Systems

Why agentic AI is a systems problem, not a model problem

> The most persistent misunderstanding in enterprise AI investment is the belief that model capability is the binding constraint. It is not — or rather, it is not the binding constraint for the vast majority of enterprises in the vast majority of use cases. The binding constraints are almost always systemic: the quality of the data the agent can reach, the reliability of the integrations it depends on, the policy infrastructure that governs what it can do, and the observability tooling that tells you whether it is doing it correctly. Understanding why agentic AI is a systems problem, and what that means for how you build and govern it, is the foundation of a readiness program that actually works.

The Model Ceiling Myth

The model improvement curve has been steep and largely unbroken since 2020. GPT-4, Claude 3, Gemini 1.5 — each generation has delivered meaningful capability gains. This has produced a pattern of organizational behavior that is understandable but counterproductive: investing heavily in model access and prompt engineering while underinvesting in everything else, on the assumption that the next model upgrade will solve whatever problems remain.

The problem with this pattern is that most agent failures in production are not caused by model inadequacy. A survey of enterprise AI program retrospectives consistently reveals a different distribution: integration failures (the tool call that returned malformed data, the API that was down, the database schema that changed without warning), data quality failures (the agent that confidently answered from stale context because the retrieval system returned an outdated document), policy gaps (the agent that took an action its operators hadn't anticipated because the permission model was underspecified), and observability gaps (no one knew the agent was making the wrong calls until the downstream effects surfaced days later).

None of these failures are solved by a better model. They are solved by better systems engineering. This is not a novel insight — every experienced architect who has moved from the demo environment to production has encountered it — but it is consistently underweighted in enterprise AI investment decisions, where model selection consumes a disproportionate share of attention and budget.

The Integration Stack

An agent's capability is, in practice, the capability of its integration stack. A highly capable model connected to a well-maintained tool catalog, a clean data layer, and a robust identity model can accomplish extraordinary things. The same model connected to brittle integrations, stale data, and an overly permissive credential store will produce errors, and those errors will be harder to diagnose precisely because the model appears confident even when it is wrong.

The integration problem has been partially addressed by the emergence of the Model Context Protocol (MCP), which Anthropic donated to the Linux Foundation in December 2025. MCP provides a standardized JSON-RPC interface between agents and the tools they use, allowing tool developers to expose capabilities through a consistent API that any compliant agent can consume. As of mid-2026, MCP has been adopted by OpenAI, Google, Microsoft, and most major enterprise software vendors. It is not a panacea — the quality of individual MCP server implementations varies widely, and a well-specified MCP interface on a poorly-maintained backend still produces unreliable tool calls — but it has substantially reduced the friction of building the integration layer.

What MCP does not address is the governance layer of integration: which tools should an agent be allowed to use, under what circumstances, with what rate limits and audit requirements? These questions are answered in the permission model and policy stack, not in the protocol itself. Organizations that have adopted MCP without also defining a tool governance policy have solved the interoperability problem while leaving the risk management problem entirely open.

Data as Infrastructure

The quality of an agent's outputs is bounded by the quality of the data it can access. This is not a new observation — it has been true of every information system ever built — but agentic AI makes it acutely visible in a new way. When a language model retrieves stale or incorrect data and produces a confident, well-written answer based on it, the error is often harder to detect than the same error produced by a conventional system, because the fluency of the output signals reliability that the underlying data does not warrant.

Retrieval-augmented generation (RAG) systems, which allow agents to retrieve relevant documents from a knowledge base at inference time, are the dominant pattern for grounding agent responses in organizational knowledge. They work well when the knowledge base is well-maintained, well-indexed, and covered by clear data governance policies. They fail in characteristic ways when the knowledge base is stale (agents answer from outdated policies or superseded pricing), when retrieval quality is poor (agents confidently answer from the wrong document), or when data classification is inadequate (agents retrieve and expose documents that should be restricted to the querying user's access level).

Data readiness — a concept explored in depth in Chapter 27 — is among the most common blockers to successful agentic deployment. Organizations that have invested in their data infrastructure, that have clean lineage, consistent classification, and reliable retrieval,

consistently achieve better agentic outcomes than those that have not, even when the latter have superior model access.

The Identity Problem

Conventional enterprise identity models were designed for humans. A person logs in, is authenticated, and receives a set of permissions scoped to their role. The session has a known owner, a bounded duration, and a human on the other end who can be held accountable for the actions taken within it. Agents break every one of these assumptions.

An agent is not a person. It may act on behalf of a person — the on-behalf-of (OBO) pattern common in OAuth — or it may act on behalf of an organizational function, with no direct human principal. It may run for extended periods, across multiple tool interactions, with a single set of credentials that accumulates an action history far more complex than any human session. It may spawn sub-agents, which inherit or derive permissions from the orchestrating agent in ways that the original credential grant did not anticipate.

The identity problem for agents is one of the less glamorous but most consequential infrastructure challenges in enterprise agentic AI. Enterprises that have solved it — that have defined agent identity as a first-class concept with its own lifecycle management, credential rotation, scope limitation, and audit trail — report significantly fewer security incidents than those that have handled agents as a special case of service account. The NIST SP 800-53 COSAiS draft overlays for single-agent and multi-agent systems, published in concept form in August 2025, provide the most detailed federal guidance on this topic to date.

Observability: The Missing Instrument

You cannot govern what you cannot observe. This truism applies to conventional software, but it applies with special force to agents, where the decision-making process is opaque, the action sequence is non-deterministic, and the failure modes are novel. An agent that is operating correctly and an agent that is heading toward a costly error can look identical from the outside — both are processing tool results and generating responses — until the error materializes.

Agentic observability requires instrumentation at three levels. Trace-level observability captures every tool call, every model invocation, every memory read and write, in a structured format that allows post-hoc reconstruction of the full decision chain. Metric-level observability captures aggregate patterns — tool call success rates, token spend per task, time to completion, escalation frequency — that allow anomaly detection and capacity planning. Evaluation-level observability runs structured automated tests against the agent's outputs to detect capability drift, policy violations, and quality degradation over time.

The tooling for agentic observability is still maturing. LangSmith, Braintrust, Honeycomb, and Arize AI have all extended their observability platforms to cover agentic traces as of 2025, but enterprise-grade capabilities — unified trace formats, cross-agent correlation, real-time alerting

on behavioral anomalies — are available only in early form. Organizations building agentic systems now should design for observability from the start, even if the tooling they deploy initially is simpler than what will be available in two years.

> *"A model is a component. An agent is a system. The governance implications of that distinction are not incremental — they are structural. Organizations that treat agent governance as a model governance problem scaled up will be consistently surprised by failures that are, in retrospect, obviously systemic."*

> CIO CHECKLIST
>
> Before approving any new agentic deployment, require answers to four infrastructure questions: What is the data freshness guarantee for the knowledge base? What is the identity model for the agent — who does it act as, and how are its credentials scoped? What observability tooling will capture its full trace, and who reviews those traces? What is the kill switch — the mechanism that stops the agent immediately if it begins behaving anomalously? A deployment that cannot answer these four questions clearly is not production-ready.

CHAPTER 05

Tool Use and Grounding

How an agent reaches into the world — APIs, retrieval, and the limits of context

> The difference between a language model and an agent is, in one narrow but important sense, a list of functions. Give a model the ability to call a search API and it can retrieve current information. Give it a code interpreter and it can run computations. Give it write access to a CRM and it can update records. Each tool added to an agent's catalog extends its reach into the world — and each extension creates both new capability and new exposure. Understanding how tool use works mechanically, what grounding actually means in practice, and where the limits of the context window create irreducible failure modes, is essential for anyone responsible for designing or governing an agentic system.

How Tool Calls Work

Modern language models support tool use through a mechanism commonly called function calling. The model is provided, at inference time, with a catalog of available tools described in a structured schema — typically a JSON specification of the function name, its parameters, and their types and descriptions. When the model determines that a tool call is appropriate, it generates a structured output specifying the tool name and parameter values. The host application executes the call, captures the result, and feeds it back into the model's context for the next inference step.

This mechanism is deceptively simple. The tool call itself is just a structured string; execution happens in application code. But the simplicity is also the source of a significant security boundary: the model cannot verify that the tool it is calling will do what its schema claims, cannot verify that the result returned to it is authentic, and cannot detect whether the environment in which it is operating has been modified to manipulate its behavior. Trust flows one way — from the application to the model — and the model has no independent verification capability.

The Model Context Protocol standardizes this exchange, providing a consistent interface between model hosts and tool servers. An MCP server exposes a set of tools through a defined protocol; an MCP client (the agent's host) discovers and invokes those tools. The protocol handles the transport and serialization details, but the trust model — which MCP servers an agent is allowed to contact, what authentication is required, what audit logging is performed — remains the application developer's responsibility. The emergence of MCP has standardized the interoperability layer; it has not standardized the governance layer.

The Grounding Problem

Grounding is the process of anchoring a model's outputs in verifiable, current, specific information rather than in parametric knowledge baked in during training. It matters because training data has a cutoff date, because it may not include proprietary or domain-specific information, and because even information that was accurate at training time may be stale at inference time. An agent that answers from parametric memory alone — without retrieving current information — is confident but potentially wrong, in ways that are difficult to detect from the quality of its prose.

The dominant grounding architecture in enterprise settings is retrieval-augmented generation (RAG): at inference time, the agent retrieves relevant documents from a knowledge base, inserts them into the context window, and generates its response against that retrieved context. RAG systems work well when retrieval quality is high — when the right documents are found, returned with high fidelity, and accurately represent current organizational knowledge. They fail in characteristic ways when any of these conditions are not met.

Retrieval failure is more common than it should be, for reasons that are primarily organizational rather than technical. Knowledge bases are often maintained with less discipline than databases: documents are added without version control, updated without invalidating the old version, or archived in formats that retrieval systems handle poorly. The agent that answers a question about the current benefits enrollment process from a document that was superseded fourteen months ago is not malfunctioning; it is functioning correctly with stale inputs. The failure is in the knowledge management system, not the model — but the failure is attributed to AI, and the trust damage accrues accordingly.

Retrieval Architectures

The engineering of RAG systems has matured substantially since the pattern was formalized in 2023. Early implementations used single-stage vector similarity search: embed the query, retrieve the most similar document chunks, insert into context. This works for many use cases but has well-documented failure modes: semantic similarity does not always align with relevance, documents that are important but not lexically or semantically close to the query may be missed, and retrieved chunks that lack surrounding context may be misinterpreted.

More sophisticated architectures use hybrid retrieval (combining vector search with lexical BM25 search), multi-stage retrieval (coarse retrieval followed by a reranking step), and structured knowledge sources (knowledge graphs, databases) alongside unstructured document stores. The choice of architecture should be driven by the information structure and the failure modes most important to avoid in the specific use case — there is no universally superior retrieval architecture, and the overhead of more complex systems is only justified when the simpler approach demonstrably fails.

Beyond retrieval, grounding can be achieved through direct tool access: a code interpreter can compute exact figures, a database tool can return exact current records, a web search tool can

retrieve current public information. Direct tool access has higher fidelity than RAG for specific factual queries, but also higher cost (in latency and token spend) and higher security surface. The combination of RAG for background context and direct tool access for specific factual queries represents the current best-practice pattern for enterprise agents that need both breadth and precision.

Context Window Limits

The context window is the agent's working memory — everything it can attend to in a single inference step. Large language models in 2025 support context windows of 128,000 to over one million tokens, depending on the model. This sounds generous, and for many use cases it is. But context windows impose costs and constraints that are easy to underestimate in design but consequential in production.

The first constraint is attention quality. Research has consistently shown that model performance degrades for information positioned in the middle of a very long context (the "lost in the middle" phenomenon), and that the most reliable positions for critical information are the beginning and end of the context. For agents that are operating with long context windows filled with retrieved documents, this means that the information the agent most needs may not be the information it attends to most reliably — a non-intuitive failure mode that requires careful prompt engineering and retrieval ordering to mitigate.

The second constraint is cost. Large context windows are expensive to process. At current pricing, a single inference call with a context window of 100,000 tokens costs orders of magnitude more than one with 1,000 tokens. For agents running multi-step loops over many iterations, the token cost of maintaining a large context through each step can overwhelm the economics of the use case. Effective agentic system design includes explicit strategies for context management: summarization of prior steps, selective retention of critical information, and periodic "context compaction" that preserves the essential state without preserving every intermediate step in full detail.

The Limits of Tool Trust

An agent is only as trustworthy as its tools. A model that reasons impeccably but is connected to a tool that returns incorrect data — whether through malfunction, misconfiguration, or deliberate manipulation — will reason correctly to an incorrect conclusion. This is the fundamental tension in agentic system design: the agent's value depends on its ability to act on tool outputs, but its safety depends on skepticism about those same outputs.

Current models do not independently verify tool outputs. They process them as authoritative by default. This is operationally convenient — verified tool calls would require a separate verification infrastructure that does not currently exist — but it creates an irreducible trust gap. Mitigations include output validation layers (post-processing tool results through rule-based checks before feeding them to the model), tool provenance logging (recording not just what a

tool returned but from which server and at what time), and explicit uncertainty signals in the tool result format (so the model can distinguish between high-confidence and low-confidence responses from the same tool). None of these mitigations eliminates the trust gap; they manage it.

> *"Grounding is not a feature you add to an agent after the fact. It is an architecture you design from the start. The agents that have surprised their operators most unpleasantly are almost always the ones where someone assumed the model would 'figure out' where to get current information."*

> PROCUREMENT NOTE
>
> When evaluating agentic platforms or frameworks, request a tool trust model document that answers: how are tool responses validated before being fed to the model? What happens when a tool returns an error or an unexpected format? What audit logging is provided for tool calls, including the full request and response? Vendors who have thought carefully about tool trust will have clear answers; those who haven't will treat these as edge cases, which is itself informative.

CHAPTER 06

Multi-Agent and Swarm

When one mind isn't enough — coordination, delegation, and emergent failure modes

> A single agent is a tool. A network of agents that coordinate, delegate, and check each other's work begins to resemble something closer to an organization — with all the efficiency gains and all the coordination costs that implies. Multi-agent architectures are not merely a scaling strategy; they are a qualitatively different mode of operation that introduces failure modes, security surfaces, and governance challenges that simply do not exist when a single agent is acting in isolation. Any enterprise planning to deploy agentic AI at serious scale will eventually encounter these architectures, and the time to understand their implications is before deployment, not after.

Why Multiple Agents

There are three primary motivations for multi-agent architectures, and they lead to different design patterns and different risk profiles. The first is specialization: some tasks are best handled by agents tuned for a specific domain or tool set, and an orchestrating agent that routes work to specialized sub-agents can outperform a single generalist agent on complex, multi-domain tasks. A research task, for example, might be handled by an orchestrator that routes to a web-retrieval agent, a document-analysis agent, and a synthesis agent — each optimized for its slice of the problem.

The second motivation is parallelism: a single agent processes steps sequentially, which is a bottleneck for tasks with parallel structure. A pipeline that collects competitive intelligence on ten companies in sequence takes ten times as long as one that dispatches ten parallel agents. The speed gains from parallelism are real and, in some use cases, decisive for the economics of the system.

The third motivation is verification: having a second agent check the work of the first is a natural quality control mechanism. The "reflection" pattern — where an agent critiques its own output, or a separate critic agent reviews the main agent's work — is among the best-validated techniques for improving agent output quality on complex tasks. Frameworks like AutoGen and CrewAI have built explicit critic and reviewer roles into their agent abstractions.

Orchestration Patterns

Multi-agent systems organize themselves around a small number of architectural patterns. The hierarchical pattern uses an orchestrator agent that decomposes tasks and delegates to sub-

agents, collecting and synthesizing their outputs. This is the dominant enterprise pattern because it maps naturally to organizational hierarchies and provides a clear point of governance: the orchestrator's behavior is the primary control surface. The peer-to-peer pattern has agents communicating laterally, without a central orchestrator. This is more resilient to orchestrator failure but harder to govern, because there is no single point where policy can be applied.

The market pattern — agents that bid for tasks, with the best-qualified agent winning the work — is an emerging research direction with interesting economic properties. The Agent2Agent (A2A) protocol, donated to the Linux Foundation by Google in June 2025, provides a standardized communication layer for peer-to-peer and market-style agent interactions. A2A complements MCP: where MCP governs agent-tool interactions, A2A governs agent-agent interactions. Together they form the interoperability layer for multi-agent enterprise systems, though both protocols are less than two years old and enterprise governance practices around them are still forming.

In practice, most enterprise multi-agent systems use a hybrid of hierarchical orchestration and peer-to-peer communication: a primary orchestrator delegates to specialist agents that communicate laterally when coordinating shared state. ServiceNow AI Agent Fabric is an example of this pattern at enterprise scale — a central fabric that manages agent registration, task routing, and observability, with individual domain agents that communicate through a shared message bus.

Emergent Failure Modes

Multi-agent systems exhibit failure modes that do not exist in single-agent systems, and that are difficult to anticipate from analysis of the individual agents in isolation. This emergence property — behaviors arising from interaction that are not present in any individual component — is one of the most challenging aspects of multi-agent governance.

The most widely documented emergent failure is cascading error propagation. When one agent in a pipeline produces an incorrect output and passes it downstream, downstream agents process the error as fact, compounding it. What begins as a small inaccuracy in the retrieval agent may become a confident, elaborately supported error by the time it reaches the synthesis agent — and the synthesis agent's confidence may make the error harder to catch, not easier. Error propagation is mitigated by inter-agent validation steps and by designing pipelines so that consequential decisions are made from multiple independent inputs rather than a single chain.

A second failure mode is goal drift in delegation. When an orchestrator delegates a sub-task to a sub-agent with an imprecise specification, the sub-agent may pursue a goal that is locally consistent with its instructions but globally inconsistent with the orchestrator's intent. This is the multi-agent equivalent of the principal-agent problem in organizational theory, and it has the same solution: clear, specific delegation with explicit success criteria and outcome verification. The difficulty is that language-based delegation is inherently imprecise, and the precision needed to prevent goal drift is higher than most orchestrator implementations currently achieve.

Security in Multi-Agent Systems

The security surface of a multi-agent system is substantially larger than that of a single agent, because every communication channel between agents is a potential attack surface. Prompt injection, which in a single-agent system requires attacking the agent's input stream, can in a multi-agent system be achieved by compromising any agent in the pipeline — including the most downstream and least scrutinized. A rogue agent — one that has been compromised through adversarial inputs or supply chain attack — can relay malicious instructions to other agents under the cover of legitimate inter-agent communication.

The OWASP Agentic Security Initiative identifies agent communication poisoning, rogue agents in multi-agent systems, and cascading hallucinations as distinct threat categories requiring specific mitigations. These mitigations include cryptographic authentication of inter-agent messages, role-based access controls applied to agent-to-agent communication, and output sandboxing that prevents sub-agents from embedding executable instructions in their responses to other agents. None of these controls are available out of the box in current multi-agent frameworks; they require explicit implementation.

The identity problem discussed in Chapter 4 is compounded in multi-agent settings. When an orchestrator delegates a task to a sub-agent, what identity does the sub-agent operate as? If it inherits the orchestrator's credentials, the attack surface is broadened — a compromised sub-agent can exercise the full permissions of the orchestrator. If each sub-agent has its own minimally-scoped credentials, the credential management complexity increases substantially. Neither approach is clearly superior; the right choice depends on the specific architecture and the relative risks of credential propagation vs. credential proliferation.

Governance at the System Level

The governance of a multi-agent system cannot be reduced to the governance of its individual agents. System-level governance requires visibility into the interactions between agents — the messages passed, the tasks delegated, the results collected — not just the inputs and outputs of each individual agent. This requires distributed tracing infrastructure that can correlate spans across multiple agents into a coherent trace of the entire pipeline's execution.

Most current governance frameworks, including the NIST AI RMF, were designed with single-system deployments in mind. Their application to multi-agent systems requires explicit extension. The Singapore Model AI Governance Framework for Agentic AI, released in January 2026, is the first governance framework to explicitly address multi-agent coordination risk as a primary concern, requiring organizations to document agent communication architectures, specify inter-agent trust models, and implement monitoring for multi-agent collusion and cascading failures.

> *"A swarm of agents is not a single agent running very fast. It is an organization — with all the coordination costs, information asymmetries, and emergent behaviors that implies. The*

> *governance framework that works for one agent is not the governance framework that works for ten agents in a network. Discovering this in production is expensive."*

ARCHITECTURE NOTE

Before deploying any multi-agent system, require a written agent communication architecture document specifying: the identity model for each agent and how credentials flow between agents, the trust model governing agent-to-agent message acceptance, the distributed tracing infrastructure, and the kill switches for individual agents and for the system as a whole. A multi-agent system without these specifications is not ready for production deployment in a consequential domain.

CHAPTER 07

The Economics of an Agent

Tokens, tools, time, and why agentic ROI looks nothing like SaaS ROI

40–70%
Cost savings at industrialized agentic deployment

12–18 mo
Typical months to reach labor cost inflection point

3–5×
Typical underestimate factor for integration engineering cost

> Every technology investment eventually comes down to a number. For agentic AI, finding that number is harder than it looks — and the models that companies have borrowed from SaaS, from robotic process automation, or from conventional cloud computing consistently produce estimates that are wrong in interesting ways. The economics of an agent are not the economics of a software license, a data center, or a human worker. They are a hybrid that shares features of all three and is reducible to none of them. Getting the unit economics right — understanding what an agent costs to run, what it returns, and how those numbers change with scale — is not an accounting exercise. It is the foundation of a credible business case, and the absence of it is the quiet reason that more enterprise AI programs stall than fail loudly.

The Cost Structure

The cost of running an agent has four main components: model inference, tool execution, memory and retrieval, and the human oversight that remains in the loop. Each behaves differently at scale, and the interactions between them are non-trivial.

Model inference cost is the most visible and most discussed. It is priced per token — typically per million tokens of input and output — and it scales with the complexity and length of the agent's reasoning. A Level 2 copilot interaction might consume a few hundred tokens. A Level 4 agent running a multi-step research task might consume hundreds of thousands of tokens across many reasoning steps, tool calls, and context updates. At the model pricing available in early 2026 — approximately $3–15 per million tokens for frontier models, and $0.10–1.00 for distilled or cached models — the inference cost per complex task ranges from fractions of a cent to several dollars depending on model choice and task design.

Tool execution costs are less discussed but can dominate. Web search API calls, database queries, code execution in sandboxed environments, and third-party API calls all carry their own pricing. A research agent that makes fifty web search calls per task is spending as much on search API calls as on model inference in some configurations. Tool execution costs are also less predictable: a model that decides to call a tool multiple times in a retry loop can generate costs

that were not in the task budget. Rate limits and explicit tool call budgets are operational controls, not just safety controls.

The ROI Model

The return side of the equation is more complex, and more debated, than the cost side. The most direct return is labor substitution: tasks previously performed by human workers, now performed by agents. But the substitution ratio is almost never one-to-one. An agent that handles routine tier-one support tickets may resolve sixty percent of them without human intervention, escalate thirty percent for human review, and handle ten percent incorrectly in ways that require remediation. The net labor saving is not sixty percent of a human support agent's time; it is sixty percent minus the supervisory and remediation overhead, which in the early phases of deployment can be substantial.

As agents mature and supervisory overhead decreases, the economics improve — sometimes dramatically. Organizations that have moved from the pilot phase to stable production with well-governed agents consistently report that the supervisory overhead drops by more than the naive learning-curve model predicts, because mature agents develop fewer novel failure modes and the support processes built around them become efficient. The inflection point — where agentic labor becomes cheaper than human labor at a given quality level — varies by use case, but it is typically reached within twelve to eighteen months of careful production deployment for well-matched use cases.

A 2025 analysis by McKinsey of 200+ enterprise AI programs found that organizations that had achieved what McKinsey terms "industrialized" agentic deployment — stable production agents at scale, with mature governance and observability — reported cost savings of 40–70% for the processes they had automated, with payback periods of twelve to twenty-four months. Organizations still in the pilot phase reported much more modest and less certain returns, reflecting the overhead-heavy early phase of the deployment curve. The gap between pilot economics and production economics is one of the most consistent findings in enterprise AI program research, and one of the most important to communicate to leadership before the program is funded.

Hidden Costs

The costs that most commonly cause enterprise AI business cases to unravel are not the model inference costs — those are well-documented and easy to estimate. They are the costs that don't appear in vendor pricing sheets: governance infrastructure, integration engineering, data quality remediation, and the organizational change management required to deploy agents in a way that the workforce accepts and the regulators tolerate.

Governance infrastructure costs — the tooling for observability, the policy engine, the audit log storage, the evaluation harness — are typically not trivial. For an enterprise deploying agents at meaningful scale, the annual cost of the governance layer can equal or exceed the annual cost

of model inference. This is a ratio that surprises many CIOs who entered the program expecting model inference to be the dominant cost.

Integration engineering — connecting the agent to the enterprise systems it needs to interact with — is frequently underestimated by a factor of three to five in initial project plans. Integration work is hard to scope before you do it, it hits organizational boundaries (the team that owns the CRM may not be motivated to prioritize the API work that enables the agent), and it often uncovers data quality problems that require remediation before the integration can be made reliable. A program that plans eighteen months and $2M for integration and discovers it needs thirty-six months and $6M is not unusual. It is merely uncomfortable.

Why Agentic ROI Looks Nothing Like SaaS ROI

SaaS investments have a recognizable ROI pattern: a licensing cost that is fixed or grows slowly with seats, a relatively rapid time-to-value (measured in weeks or months rather than years), and a value curve that is relatively flat once the product is deployed — there are no dramatic improvements in the product's output quality over time unless the vendor ships a new version. Agentic AI ROI has none of these characteristics.

The cost structure is variable and usage-driven, not fixed. The time-to-value is longer, because the integration, data preparation, governance, and evaluation work extends the ramp. But the value curve, once the inflection point is reached, can be substantially steeper: an agent whose quality and reliability is improving over time, in a process that is being continuously refined by an evaluation harness, can deliver compounding improvement in ways that a conventional SaaS application cannot. The enterprise that builds a high-quality agentic capability in year one is not buying a static product; it is building an asset whose value grows with investment.

This distinction matters for how the investment is governed. SaaS investments are largely recurring OpEx; the value is realized continuously and the decision to continue is re-evaluated annually. Agentic AI investments have a more complex profile: high upfront costs (integration, governance, evaluation infrastructure), a period of negative or uncertain return, followed by a period of strong return that increases with scale. This profile is more like a product development investment than a software license, and it should be budgeted and governed accordingly — with a portfolio approach that accepts early-stage uncertainty in exchange for later-stage value, rather than a line-item approach that requires positive ROI in year one.

> *"The business case for agentic AI that goes wrong almost always has the same flaw: it compares the cost of the agent to the cost of the human at the task level, and misses the cost of the system that supports the agent. That system — governance, integration, data, observability — is what makes the agent reliable, and without reliability, the task-level economics are meaningless."*

Pricing the Agent Portfolio

The unit economics question — what does it cost to run this agent on this task? — is answerable with reasonable precision once an agent is in stable production and the costs of all layers are measured. Before that point, estimates require explicit assumptions about supervisory overhead, integration cost amortization, and governance infrastructure allocation. Making those assumptions explicit — and tracking actuals against them — is the financial management discipline that separates successful agentic programs from those that run out of budget before reaching the inflection point.

A portfolio approach to agent investment — analogous to a product portfolio rather than a project portfolio — is the governance structure that most consistently produces good outcomes. Treat each agent as a product with a lifecycle: seed investment for development and integration, a validation phase where quality and economics are measured, a production phase where the investment is scaled based on demonstrated returns. Kill agents that fail to reach the validation threshold rather than continuing to invest in hope. Fund the ones that pass at the level their demonstrated returns justify.

> CFO NOTE
>
> When reviewing an agentic AI business case, require five numbers: model inference cost per task (estimated from a pilot), tool execution cost per task, integration engineering cost (total, amortized over a three-year deployment), governance infrastructure cost (annual), and supervisory overhead cost at projected scale. A business case that can provide all five numbers, with supporting assumptions, is credible. One that can only provide the first is guessing at the others.

CHAPTER 08

Failure Modes

Hallucinated calls, prompt injection, runaway loops, and the cost of being wrong at scale

> Every technology has its characteristic failure patterns. The failure patterns of agentic AI are not random — they follow from the architecture, the trust model, and the autonomy level of the system. Understanding them in advance is not pessimism; it is engineering discipline. The organizations that have deployed agents most successfully are those that anticipated the specific failure modes most likely to affect their systems and built explicit mitigations before the failures occurred. The ones that have had the most costly incidents are those that treated failure as an unlikely edge case rather than a predictable engineering challenge.

Hallucinated Tool Calls

The most operationally significant failure mode in production agentic systems is the hallucinated or malformed tool call: the agent generates a tool call with incorrect parameters, a non-existent function name, or a parameter value that is internally inconsistent with the task context. In a well-designed system, such calls are caught by the tool-call validation layer and returned to the agent as errors. In a poorly designed system — or when the validation is incomplete — they may be silently dropped, produce unexpected side effects, or propagate errors into downstream processing.

Hallucinated tool calls are distinct from hallucinated content (the fabrication of facts in text output) but they arise from the same underlying mechanism: the model generating plausible-sounding output that is not grounded in verified information. The mitigation strategies are also analogous: strict schema validation for all tool calls, explicit error handling for malformed calls that feeds back into the model's context rather than silently failing, and evaluation harnesses that specifically test for tool-call reliability across the range of inputs the agent is expected to encounter.

The frequency of hallucinated tool calls varies substantially between models and between task types. Tasks that involve calling well-known, frequently-used APIs — web search, calendar access — have lower hallucination rates than tasks that involve novel or complex API schemas. Organizations deploying agents with large tool catalogs should expect higher hallucination rates than those with small, well-defined catalogs, and should design their evaluation frameworks accordingly.

Prompt Injection

Prompt injection is the agentic equivalent of SQL injection: adversarial content embedded in the agent's input stream that hijacks its behavior. In the simplest form — direct prompt injection — a user provides input that contains instructions designed to override the agent's system prompt. In the more dangerous form — indirect prompt injection — the adversarial content arrives through a tool call: in a document the agent retrieves, a web page it visits, a database record it queries, or an email it reads.

Indirect prompt injection is more dangerous than direct injection for two reasons. First, the content arrives from a source that the agent is designed to trust — a retrieved document, a tool result — and may not be treated with the same skepticism that an agent (or its operators) would apply to user-provided input. Second, because the agent is autonomous, the adversarial instruction can direct it to take actions — send a message, exfiltrate data, modify a record — without any further human interaction. The attack requires no social engineering of the human; it only needs to reach the agent's context window.

Documented indirect prompt injection attacks have been demonstrated against multiple production agentic systems, including browser-use agents and email-processing agents. The MITRE ATLAS knowledge base catalogs the techniques involved; the OWASP LLM Top 10 ranks prompt injection as the top risk for LLM applications (LLM01). Mitigations include input/output sandboxing, instruction hierarchy enforcement (instructions from the system prompt take precedence over instructions from retrieved content), and explicit validation of agent actions against the original task goal before execution.

Runaway Loops

A runaway loop is an agent that continues executing indefinitely — consuming tokens, tool calls, and time — without making progress toward its goal or reaching a completion condition. Loops arise from several distinct mechanisms: a goal that cannot be met with the available tools (the agent retries indefinitely), a planning error that places the agent in a circular sequence of steps, or a tool failure that the agent attempts to recover from but cannot. The result in all cases is consumption of resources and, often, an accumulating set of side effects from the failed recovery attempts.

Runaway loops are the failure mode with the clearest mitigation: explicit resource budgets. Every production agent should have a hard limit on the number of tool calls it can make in a single session, the total number of tokens it can consume, and the elapsed wall-clock time before it is halted and the situation flagged for human review. These limits should be tuned to the expected execution profile of the agent's tasks — tight enough to catch runaway behavior early, loose enough not to interrupt normal operation — and they should be monitored to detect drift in the agent's execution patterns over time.

Beyond resource limits, graceful degradation matters. An agent that hits its resource limit should not simply stop; it should attempt to preserve whatever partial work it has completed,

communicate the state of the task to the human operator, and explain, to the extent it can, why it was unable to complete. The quality of an agent's failure communication is a meaningful differentiator between well-engineered and poorly-engineered systems, and it is underspecified in almost all current vendor documentation.

Scope Creep and Overreach

Scope creep is the gradual expansion of an agent's operational footprint beyond what was intended. It happens for several reasons: agents instructed to "do whatever it takes" to complete a task will, if given sufficient tool access, eventually access resources and take actions that were not explicitly sanctioned; agents operating in multi-step tasks may acquire context that makes additional actions seem locally justified even when they are globally outside the intended scope; and agents that accumulate permissions through on-behalf-of delegation chains may end up with effective access rights far beyond those of the immediate delegating user.

The OWASP LLM Top 10 designates this failure mode "Excessive Agency" (LLM06) and identifies over-permissioning — granting agents more tool access, permissions, or capabilities than the minimum required — as the primary structural cause. The mitigation is principled: start with minimal permissions, grant additional access only when there is a demonstrated requirement, and review the permission envelope periodically as the agent's operational scope is better understood. This principle is easy to state and persistently difficult to implement in organizations where the path of least resistance is to give the agent broad permissions and rely on its judgment to use them appropriately.

The Failure Feedback Loop

Perhaps the most important meta-level failure mode is the absence of a failure feedback loop: the organizational condition in which agent failures occur, produce costs or user harm, and are never routed back to the teams responsible for the agent in a form that enables learning and remediation. This failure mode does not have a technical name in the OWASP taxonomy, but it is arguably more consequential than any specific technical failure mode, because it allows all the others to persist.

Healthy failure feedback loops require incident reporting that reaches the agent development team, retrospective analysis that distinguishes failure types and root causes, evaluation updates that add new test cases for each failure mode discovered, and governance processes that track remediation of known failure modes. Organizations with mature DevOps practices will find these requirements familiar; the challenge is applying them to agents, where the failure modes are novel, the evidence (a reasoning trace) is less familiar than a stack trace, and the operational responsibility may not yet be clearly assigned.

> *"The three most expensive agent failures I have encountered in enterprise settings had one thing in common: they were not surprising to anyone who had done the pre-deployment risk analysis.*

> *They were failures that had been identified as risks and classified as unlikely edge cases. The edge cases turned out to be more frequent than estimated."*

> QA CHECKLIST
>
> Before deploying any agent into production, run explicit tests for the four canonical failure modes: hallucinated tool calls (use adversarial inputs that are likely to produce malformed calls), prompt injection (embed adversarial instructions in the test retrieval content), resource budget exhaustion (run the agent on tasks that are likely to require many steps), and scope overreach (run the agent with a task framed to encourage extra-scope actions and verify that permissions prevent them). Document the results and the mitigations applied.

CHAPTER 09

The Threat Landscape

OWASP Agentic AI Top 10, MITRE ATLAS, and adversaries who know agents better than you do

15
OWASP Agentic AI threat categories (ASI v1.0)

167
MITRE ATLAS techniques as of 2025

57
MITRE ATLAS documented case studies

> Agentic AI does not face a new kind of adversary. It faces familiar adversaries — financially motivated criminals, nation-state actors, opportunistic hackers — who have learned to exploit a new attack surface with significant speed. The tools of classical cybersecurity apply: threat modeling, layered defense, incident response. What is new is the specific texture of the vulnerabilities that agents introduce, and the speed at which adversaries have developed techniques to exploit them. Several documented attack campaigns against production agentic systems predate the governance frameworks designed to address them. Understanding the threat landscape, in its current specificity, is the prerequisite for building a defense that is more than decorative.

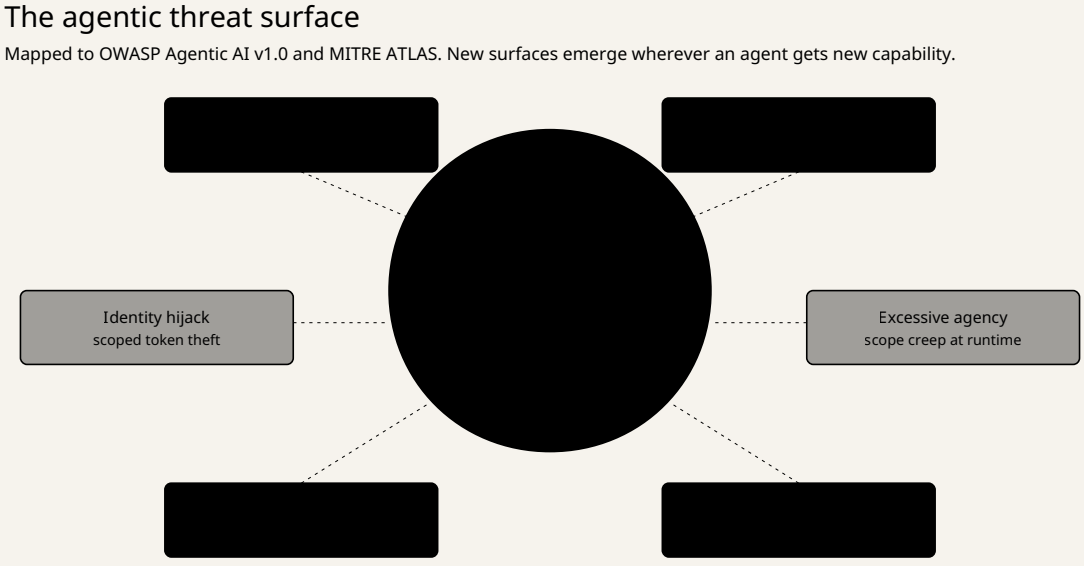

FIGURE 9.1SIX CATEGORIES OF AGENT-SPECIFIC THREAT. MOST PREDATE AGENTIC AI; AGENCY JUST MAKES THEM MORE EXPENSIVE.

The OWASP Taxonomy

The most comprehensive published taxonomy of agentic AI threats is the OWASP Agentic Security Initiative's Threats and Mitigations document, version 1.0, published April 2025. It identifies fifteen threat categories purpose-built for autonomous agent systems, organized by attack surface and impact type. The OWASP LLM Top 10 v2025 provides the broader LLM application context; the ASI taxonomy provides the agentic-specific extension.

The most operationally significant categories for enterprise deployments are: **Memory Poisoning** — injecting false data into an agent's persistent memory store to influence future behavior across sessions; **Tool Misuse** — manipulating an agent through its tool inputs to abuse tool access for unauthorized purposes; **Privilege Compromise** — exploiting weaknesses in the agent's permission management to acquire capabilities beyond its intended scope; **Cascading Hallucinations** — exploiting error propagation in multi-agent pipelines to amplify a small initial manipulation into a large downstream impact; and **Agent Communication Poisoning** — manipulating the messages passed between agents in a multi-agent system.

Less discussed but increasingly relevant is **Human Manipulation**: using an agent that has established implicit trust with a user to coerce or deceive that user in ways that the agent's operators did not intend and cannot easily detect. An agent that has managed a user's calendar and email for several months has accumulated significant implicit trust. An adversary who can influence that agent's behavior — through memory poisoning or prompt injection — can leverage that trust for social engineering at scale.

MITRE ATLAS

The MITRE ATLAS knowledge base — Adversarial Threat Landscape for AI Systems — provides the ATT&CK-style structured taxonomy that security teams need to build threat models and red team exercises. As of 2025, ATLAS documents 16 tactics, 167 techniques, and 57 case studies of real-world AI attacks. The case studies are particularly valuable: they ground abstract techniques in documented incidents, making the threat model actionable rather than theoretical.

ATLAS coverage of agentic AI attack surfaces is actively expanding. The most relevant technique groups for enterprise agentic deployments include: LLM Prompt Injection (AML.T0051) and its sub-techniques covering direct, indirect, and stored injection; Backdoor ML Model (AML.T0020), relevant for supply chain attacks against third-party model providers; and Context Manipulation techniques that exploit the agent's trust in retrieved content. ATLAS integrates with OWASP LLM Top 10 via cross-referencing technique identifiers, enabling security teams that are familiar with one framework to orient in the other.

The MITRE AI Incident Sharing Initiative, launched alongside ATLAS, allows organizations to report and share AI security incidents in a standardized format. Early-stage enterprise AI programs have a strong interest in reading these reports — they represent some of the most specific threat intelligence available on real-world agentic attacks — and a long-term interest in contributing, both to build collective defense and to receive curated intelligence in return.

Supply Chain Threats

An agent is only as trustworthy as every component in its supply chain. The supply chain for a modern enterprise agent includes: the foundation model (from a vendor), the orchestration framework (typically open-source), the MCP servers providing tool access (from a mix of vendors and internal developers), the vector store and retrieval system (from a vendor), the evaluation framework (open-source or proprietary), and the underlying cloud infrastructure. Each component is a potential attack surface, and the attack surface of the assembled system is substantially larger than the sum of its parts.

Malicious MCP servers — packages that present as legitimate tool providers but exfiltrate data, inject instructions into tool responses, or manipulate the agent's tool call parameters — represent an emerging supply chain threat that has no direct analogue in conventional software. An organization that deploys an agent with an MCP server from a third party without auditing that server's behavior is accepting a supply chain risk that may be invisible until the attack has already succeeded. The practice of "MCP security review" — auditing third-party MCP server implementations before deploying them in production — is a nascent but necessary discipline.

Model provider supply chain risk is a different but equally real concern. Enterprises that deploy agents built on third-party foundation models are dependent on those providers' security practices for the integrity of the model's behavior. Vendor Responsible Scaling Policies, such as Anthropic's RSP, provide some visibility into provider security practices — they specify the testing and safety evaluations conducted before model deployment — but they do not eliminate the dependency. Organizations deploying agents in high-risk domains should require vendor security attestations and, where possible, evaluate model behavior against known adversarial test cases before production deployment.

The Adversary Advantage

One of the less comfortable facts in the agentic threat landscape is that adversaries have, in several documented cases, developed and deployed attack techniques against agentic systems before the defending organizations had completed their threat models. This is not unusual in cybersecurity — the attacker advantage in time-to-exploit is well-documented — but the agentic surface presents some specific characteristics that accelerate adversary learning.

Agents are, at least partially, transparent about their capabilities: they tell users what tools they have, what they can access, and how they process requests. This transparency, designed to build user trust, also builds adversary knowledge. An attacker who can interact with an agent has a detailed map of its attack surface. The techniques for exploiting that surface — particularly prompt injection and tool misuse — are well-documented in open literature and require no specialized capabilities to attempt. The barrier to entry for agentic AI attacks is lower than for most classes of cyberattack.

Defense in Depth

The security architecture for enterprise agentic systems should follow the same defense-in-depth principles that govern conventional cybersecurity, with layers adapted to the specific threat surface. At the input layer: prompt injection detection, content classification for retrieved documents, and explicit validation of tool call parameters. At the execution layer: least-privilege tool access, rate limiting, and behavioral anomaly detection. At the output layer: output sandboxing, post-processing validation, and human review for high-consequence actions. At the infrastructure layer: secure MCP server deployment, model provider attestations, and audit logging with integrity guarantees.

No single control is sufficient. The depth is the point: an attacker who defeats one layer should encounter the next. The specific controls most important for a given deployment depend on the threat model — which adversaries are realistic, which attack vectors are accessible, which assets are most valuable to protect — and that threat model should be built explicitly, using OWASP ASI and MITRE ATLAS as structured inputs, before deployment rather than after the first incident.

> *"The agentic attack surface is not theoretical. The techniques in OWASP ASI and MITRE ATLAS are documented from real incidents against real production systems. The organizations that treat these as research documents rather than operational threat intelligence are making a choice about risk that they may not have made explicitly."*

> CISO PRIORITY LIST
>
> Three controls to implement before any production agentic deployment: (1) indirect prompt injection detection — every retrieved document, tool response, and external input should be processed through an instruction-detection layer before being inserted into the agent's context; (2) tool call logging with integrity guarantees — every tool call, its parameters, and its result should be logged in a tamper-evident store, not just for post-incident analysis but for real-time anomaly detection; (3) kill switches at multiple levels — individual tool disablement, individual agent halt, and full-system shutdown, each testable independently.

CHAPTER 10

The Regulatory Perimeter

EU AI Act, US executive orders, sectoral rules — what bites and what doesn't

€35M *or 7% of global turnover*
Maximum EU AI Act penalty for prohibited practice violations

Aug 2026
EU AI Act full enforcement date for high-risk AI systems

> The regulatory landscape for agentic AI is neither as clear as regulators claim nor as murky as vendors prefer. Several major frameworks are now in effect, more are in enforcement phases that will become material within the next two years, and the sectoral overlays — in financial services, healthcare, critical infrastructure, and defense — add specificity that general AI regulations lack. Understanding the regulatory perimeter is not optional for enterprises deploying agents in consequential domains. The question is not whether regulation applies; it is which regulations apply, with what obligations, on what timeline, and with what penalties for non-compliance.

The EU AI Act

The EU AI Act (Regulation 2024/1689) entered into force on August 1, 2024, making it the world's first comprehensive, binding AI regulation with significant extraterritorial reach. Any enterprise deploying AI systems that affect EU residents — regardless of where the deploying organization is headquartered — is within scope. The regulation's enforcement is phased: prohibited practices (social scoring, subliminal manipulation, real-time remote biometric ID in public spaces) became enforceable in February 2025; high-risk AI obligations (Annex III systems in employment, education, credit, law enforcement, and others) become fully enforceable in August 2026; and General Purpose AI Model obligations were effective from August 2025.

Agentic AI sits awkwardly in the EU AI Act's risk taxonomy. The regulation does not use the term "agentic AI" and was drafted primarily with static AI systems in mind. However, several provisions apply directly to autonomous agents: GPAI models with "autonomous scalability and tool access" are flagged for systemic risk consideration; systems that operate with meaningful autonomy in high-risk domains fall under Annex III obligations regardless of their autonomous character; and the human oversight requirements in Article 14 apply to any high-risk AI system, which agentic deployments in regulated domains will typically be. The EU AI Office has stated that clarification guidance on agentic AI classification will be a priority in 2026.

For enterprises, the practical AI Act obligations for likely high-risk agentic deployments include: a risk management system (Article 9) documenting the agent's operation and residual risks; data

governance practices (Article 10) covering the training data and, implicitly, the retrieval data used by RAG systems; technical documentation (Article 11) that can survive a regulatory audit; logging requirements (Article 12) that create a record of the agent's operations; and human oversight mechanisms (Article 14) that are genuinely effective, not merely nominal. The penalties for non-compliance — up to €35 million or 7% of global annual turnover for the most serious violations — are large enough that compliance cannot be treated as optional by any organization of scale.

The US Federal Landscape

The United States federal AI regulatory landscape has been in flux since the Biden administration's Executive Order 14110 on AI was revoked by the Trump administration's Executive Order 14179 in January 2025. EO 14179 takes a markedly different posture: innovation-first, with less emphasis on pre-deployment safety requirements. However, the federal AI governance infrastructure built under EO 14110 — including NIST's AI Risk Management Framework and the sector-specific AI frameworks developed by agencies — has not been dismantled and continues to operate.

The Office of Management and Budget's M-25-21, issued in April 2025, replaced Biden-era M-24-10 and established the current federal AI governance requirements: Chief AI Officers at major agencies, AI impact assessments for high-impact uses, and human oversight for AI decisions that affect individuals' significant rights and interests. M-25-21 is binding only for federal agencies, but it heavily influences the expectations of federal contractors and regulated industries. Enterprises in defense, healthcare (CMS, FDA-regulated devices), financial services (OCC, Federal Reserve), and critical infrastructure should treat federal agency guidance as leading indicators of mandatory requirements for their own sectors.

At the state level, Colorado's AI Act (effective February 2026) established the first US state-level high-risk AI obligations broadly analogous to the EU AI Act's framework. California's AI transparency and safety bills represent a second major jurisdiction with emerging requirements. The patchwork nature of US state AI regulation — and the prospect of a federal preemption debate — creates planning uncertainty that is best addressed by building toward the most demanding applicable standard rather than optimizing for the most permissive.

Sectoral Rules

General AI frameworks set the floor; sectoral rules add the ceiling in specific industries. In financial services, the US banking regulators' revised model risk guidance (OMB-led revision, finalized April 2026) explicitly excludes generative and agentic AI from its scope — not as an exemption but because regulators have signaled they will issue separate, more specific guidance for AI-based decision systems. The OCC has published supervisory letters on AI governance that apply to agents used in credit underwriting, fraud detection, and customer

communications. MiFID II and DORA in the EU impose specific requirements for operational resilience and outsourcing that apply to AI systems used in financial services.

In healthcare, the FDA's AI-Enabled Device Software Functions framework (finalized guidance December 2024) applies to AI systems used in clinical decision support and medical devices. Agentic AI deployed in clinical settings — for patient record review, diagnostic support, treatment recommendation — may qualify as a Software as a Medical Device (SaMD) and require pre-market submission. The FDA's Predetermined Change Control Plan requirements, which allow for AI systems that update based on real-world performance without re-submission, are directly relevant to agentic systems whose behavior may evolve over time.

In critical infrastructure — energy, water, transportation, telecommunications — CISA and sector-specific regulators have published AI risk guidance that, while not yet binding for most organizations, signals the direction of mandatory requirements. The EU NIS2 directive, which became effective in October 2024 and is being transposed into member state law through 2025, imposes cybersecurity requirements for AI systems used in essential services that subsume agentic AI deployments in regulated infrastructure.

The Accountability Gap

One of the most persistent regulatory tensions in agentic AI is the accountability gap: the difficulty of assigning clear legal and regulatory responsibility for actions taken by an autonomous system with multiple principals — the model provider, the platform deployer, the enterprise deployer, the end user. The EU AI Act's provider/deployer distinction provides a starting point: the provider (who places the AI system on the market) bears obligations for system design and documentation; the deployer (who deploys it in a specific context) bears obligations for appropriate use and human oversight.

But in an agentic supply chain — where a foundation model is trained by one company, integrated into an orchestration framework by a second, deployed on a cloud platform by a third, fine-tuned by a fourth, and operated by an enterprise — the provider/deployer distinction becomes a complex, contested, and potentially litigation-rich question. Organizations should document their position in the AI supply chain for each agentic deployment, identify which obligations apply to them as providers vs. deployers vs. users, and obtain contractual representations from upstream providers about their compliance with applicable obligations. This is unglamorous legal work, but it is the work that determines who answers when something goes wrong at scale.

Preparing for the Next Regulation

The regulatory landscape for agentic AI is in its early innings. The EU AI Act is the most mature framework, but it is not yet fully enforced. The US federal landscape will evolve as specific AI-related incidents drive regulatory responses. The OECD AI Principles, updated in 2024 to reflect generative and autonomous AI, continue to influence national legislation across forty-two

signatory countries. Organizations that build toward the highest common denominator — full EU AI Act compliance, NIST AI RMF alignment, and sector-specific requirements — will spend more upfront but avoid the more expensive retrofits that selective compliance strategies typically require.

The OECD AI Principles provide a useful international convergence reference: their five principles — inclusive growth, human-centred values, transparency, robustness and security, accountability — are reflected in virtually every national AI regulatory framework. Organizations whose governance programs demonstrably embody these principles will find regulatory alignment easier and more credible than those whose compliance is narrowly legalistic.

> *"The regulators who are writing today's AI rules are largely doing so without having deployed an agent. The rules that emerge from that process will be imperfect. But they will be enforceable. The enterprise that waits for perfect rules before building governance infrastructure will wait forever and comply never."*

> LEGAL COUNSEL NOTE
>
> For each agentic deployment in scope of the EU AI Act, prepare a one-page risk classification memo documenting: the system's risk tier under Articles 6–9 and Annex III, the organization's role as provider vs. deployer, the human oversight mechanism and its adequacy assessment, and the logging and audit capability. This memo is both a governance discipline and, increasingly, the first document a regulator will request in a supervisory inquiry.

CHAPTER 11

The Frameworks Landscape

NIST, ISO, OECD, OWASP, the analyst maturity models — ranked by what actually moves the needle

48+
Published AI governance frameworks, standards, and guidelines (2026)

Jan 2026
Singapore MGF for Agentic AI published — first agentic-specific governance framework

15
OWASP Agentic AI threat categories (v1.0, April 2025)

> The frameworks landscape for agentic AI governance is, in mid-2026, simultaneously overcrowded and underbuilt. There are more than forty published frameworks, standards, guidelines, maturity models, and codes of practice claiming relevance to enterprise AI governance. Most of them were written before agentic systems existed at scale. Several are excellent for the problem they were designed for and deeply inadequate for the problem enterprises now face. A handful are genuinely essential. Navigating this landscape efficiently — knowing which frameworks to adopt, in what combination, with what investment — is one of the highest-leverage decisions in an enterprise agentic readiness program.

The Mandatory Baseline

Three frameworks constitute the mandatory baseline for any enterprise operating at scale with agentic AI. They are mandatory not because they are the most technically sophisticated but because they are the ones that regulators, auditors, procurement officers, and boards will ask about — and the ones whose absence will create a defensibility problem in any governance conversation.

The NIST AI Risk Management Framework (AI 100-1, January 2023) is the de facto governance vocabulary for AI risk management in the United States and increasingly globally. Its four functions — Govern, Map, Measure, Manage — provide the structural skeleton around which an enterprise AI governance program should be built. Its companion document, NIST AI 600-1 (the Generative AI Profile, July 2024), maps 200+ suggested actions to LLM-specific risk areas. The NIST AI RMF's limitation — what the Cloud Security Alliance has termed the "agentic fitness gap" — is that neither document was written with autonomous agents in mind. The GOVERN function lacks autonomy tier concepts; the MAP function stops at the model boundary; AI 600-1 is content-generation–centric. NIST has acknowledged these gaps with its February 2026 AI Agent Standards Initiative, with an AI Agent Interoperability Profile planned for Q4 2026. Until

that profile arrives, the CSA's proposed Agentic Profile extension provides the most rigorous bridge.

The EU AI Act is, for any enterprise serving EU markets, the highest-stakes mandatory framework. Its phased enforcement creates a false sense of runway; the technical documentation, risk management, and human oversight obligations of Articles 9–14 require organizational readiness that takes twelve to eighteen months to build from scratch. Enterprises that have not begun their AI Act compliance programs should treat August 2026 — the high-risk enforcement date — as a hard deadline for a program that should have started in early 2025. The Act's extraterritorial reach means that headquarters location is irrelevant; what matters is whether the AI system affects EU residents.

ISO/IEC 42001:2023 is the first certifiable AI management system standard, and it is becoming a procurement requirement at the same pace that ISO 27001 became a procurement requirement for cybersecurity. Its Plan-Do-Check-Act structure integrates naturally with existing GRC programs; organizations that have already achieved ISO 27001 certification will find 42001 familiar and incrementally achievable. Like the NIST AI RMF, 42001 provides an organizational governance shell rather than specific technical controls for autonomous agents — the standard does not tell you what to audit in an agentic deployment, only that you should have a management system. The gap must be filled by more technically specific frameworks.

The Operational Layer: Singapore and OWASP

The mandatory baseline frameworks tell organizations how to organize their governance programs. The operational layer frameworks tell them what, specifically, to govern in agentic deployments. Two frameworks stand out as essential for this purpose.

The Singapore Model AI Governance Framework for Agentic AI, published in January 2026, is the world's first governance framework dedicated specifically to autonomous AI agents. Its four governance dimensions — risk assessment and bounding, human oversight checkpoints, technical safeguards, and transparency and user education — map precisely to the operational challenges of enterprise agentic deployment. It requires organizations to inventory deployed agents, define per-agent safety boundaries, implement human-override mechanisms proportional to risk, and maintain end-to-end audit trails for agent decisions and actions. The framework's voluntary status does not diminish its operational value; it is the most directly actionable governance template available for agentic deployments, and it is already influencing regulatory expectations across ASEAN and beyond.

The OWASP Agentic AI Threats and Mitigations document (v1.0, April 2025) provides the threat taxonomy that the Singapore framework's technical safeguard requirements implicitly reference. Its fifteen threat categories — memory poisoning, tool misuse, privilege compromise, cascading hallucinations, rogue agents, agent communication poisoning, human manipulation, goal hijacking, and others — are the specific threat classes that security controls must address. Used in conjunction with the OWASP LLM Top 10 v2025 (which covers the broader LLM application context, including Excessive Agency at LLM06), OWASP ASI provides the most complete

security threat model available for enterprise agentic systems. It is free, practitioner-oriented, and continuously updated — the combination of attributes that made OWASP's security work indispensable in web application security applies equally here.

The Gartner AI TRiSM framework — AI Trust, Risk, and Security Management — serves a different function from the technical frameworks above. Its four pillars (AI Governance, Explainability, ModelOps, Application Security) are most useful as an executive communication framework and a vendor evaluation lens. The TRiSM label has driven a generation of AI governance tooling; the Market Guide format helps organizations understand the vendor landscape. The 2025 TRiSM update explicitly addresses agentic AI governance requirements. TRiSM is most valuable not as a primary implementation guide but as the language for communicating AI governance investments to boards and procurement teams who are already familiar with Gartner's framing.

A-Tier: High Value, Specific Applications

Several A-tier frameworks deliver significant value for specific applications within the enterprise agentic governance program. MITRE ATLAS is essential for security teams building threat models and red team exercises; its ATT&CK-style structure is familiar and its case studies ground abstract techniques in reality. The CSA AI Controls Matrix (AICM, July 2025) provides 243 control objectives across eighteen security domains, mapping to NIST AI RMF, ISO 42001, and the EU AI Act — a valuable cross-referencing resource for organizations that need to demonstrate compliance with multiple frameworks simultaneously.

The vendor Responsible Scaling Policies — Anthropic's RSP, Google DeepMind's Frontier Safety Framework, OpenAI's Preparedness Framework — are relevant to enterprise procurement risk assessment. They specify the capability thresholds at which providers will pause deployment, the safety evaluations conducted before model release, and the escalation mechanisms for safety incidents. Reading them before signing an enterprise model contract is, in the current environment, a reasonable due diligence standard. The Model Context Protocol and Agent2Agent protocol are the interoperability standards that enterprise agentic architectures will be built around; understanding their governance implications — tool trust models, agent identity frameworks, audit trail requirements — is an architectural prerequisite, not an optional enrichment.

For organizations that need a maturity-based diagnostic — a way to assess where they are and what good looks like — the analyst maturity models serve well. The MIT CISR four-stage model (Experiment → Build Pilots → Industrialize → Future-Ready), Gartner's five-level Agentic AI Maturity Roadmap, and McKinsey's agentic mesh architecture framework are grounded in research and enterprise data respectively, and their diagnostic questions are useful for identifying gaps and sequencing investments. None of them substitute for the operational frameworks above, but they provide the vocabulary for strategy conversations that the technical frameworks do not.

What Not to Do

The most common mistake in the frameworks landscape is attempting to adopt too many of them simultaneously. An organization that tries to map its governance program to NIST AI RMF, ISO 42001, EU AI Act, Singapore MGF, OWASP ASI, OWASP LLM Top 10, MITRE ATLAS, CSA AICM, and three analyst frameworks at the same time will produce an elaborate mapping document and very little operational governance. Framework fatigue is real, and it is a significant cause of governance programs that look impressive on paper and accomplish nothing in practice.

The practical approach is a structured stack: NIST AI RMF + ISO 42001 + EU AI Act as the mandatory baseline, adopted in that order of operational priority (NIST for program structure, ISO 42001 for certification readiness, EU AI Act for compliance deadlines). Singapore MGF for Agentic AI + OWASP ASI as the operational layer, applied to each specific agentic deployment. MITRE ATLAS for security threat modeling. Analyst maturity models for diagnostic and communication purposes. Additional frameworks added only when a specific gap — a sectoral regulatory requirement, a procurement qualification, a specific technical control need — is identified that the stack does not address.

> *"A governance program with three frameworks deeply understood and operationally implemented is worth more than one with fifteen frameworks superficially mapped. The frameworks are not the governance. They are the vocabulary in which the governance is written. The governance is the people, the processes, and the decisions."*

The Ranking Conclusion

The clearest guidance that emerges from a careful reading of the full frameworks landscape is this: NIST AI RMF plus ISO/IEC 42001 plus the EU AI Act constitute the mandatory compliance and governance baseline — they are the frameworks every enterprise operating at scale must engage with, regardless of geography or sector. The Singapore Model AI Governance Framework for Agentic AI plus OWASP Agentic AI represent the operational layer — the frameworks that tell you what to actually do when deploying an agent, not just how to organize the program. The analyst maturity models — Gartner's Agentic AI Maturity Roadmap, McKinsey's architecture framework, MIT CISR's four-stage model — serve the diagnostic and communication function, helping organizations understand where they are and explaining it to stakeholders. Everything else is supplementary. Start with the first three, operationalize the next two, use the last three for diagnosis and communication, and add further frameworks only when a specific identified gap requires them.

PROGRAM OFFICE NOTE

A practical approach to the frameworks landscape: build a single master control matrix with rows for governance controls (drawn from the NIST AI RMF and ISO 42001 requirements) and columns for each relevant framework (EU AI Act, Singapore MGF, OWASP ASI, sector-specific rules). Map each control once; demonstrate compliance with multiple frameworks through a single evidence set. This approach reduces documentation burden by 40–60% relative to maintaining separate compliance programs for each framework, and it surfaces gaps more clearly than framework-by-framework analysis.

CHAPTER 12

The Four Pillars

A readiness model: governance, orchestration, use case identification, integration

> Every assessment framework eventually converges on a question of structure: what are the distinct capabilities that an organization must develop, and how do they relate to each other? This report's answer, derived from analysis of dozens of enterprise agentic deployments across industries and from the governance frameworks examined in the preceding chapter, is four pillars. Governance and risk. Orchestration and architecture. Use case identification and portfolio management. Integration and data. An enterprise can be strong in three and critically weak in the fourth, and the weak pillar will determine the program's outcomes regardless of the others' strength. Understanding the four pillars — what each encompasses, why it matters, and how it interacts with the others — is the bridge from the conceptual foundation built in Part I to the operational detail of Part II.

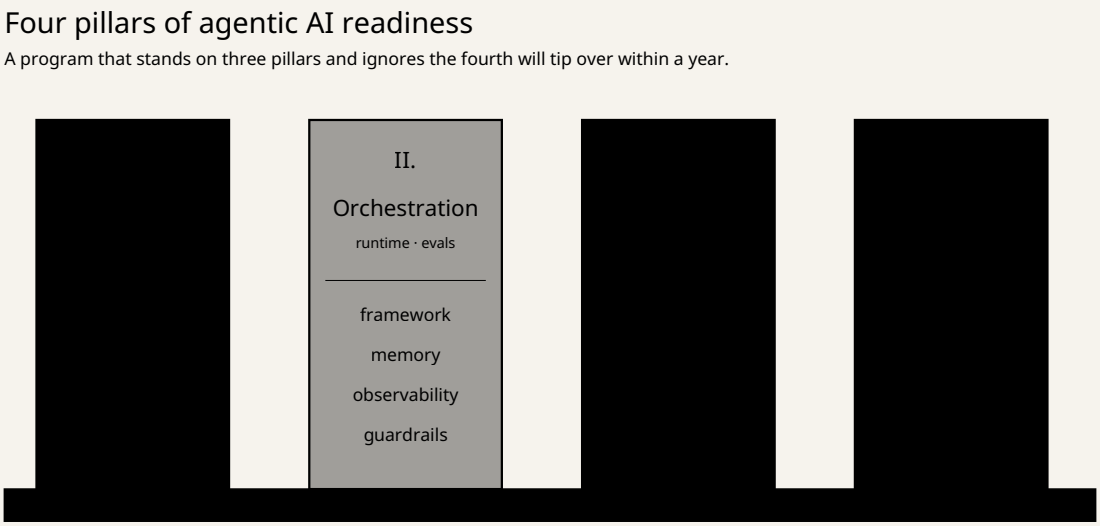

FIGURE 12.1THE FOUR PILLARS. EACH RESTS ON THE SAME FOUNDATION: ENTERPRISE MISSION, RISK APPETITE, AND CULTURE.

Pillar I — Governance and Risk

Governance is the first pillar, and it is first not because it is the most technically interesting but because it is the one that enables the others. Without a governance framework — a clear assignment of accountability, a policy stack that translates risk appetite into operational rules, a risk management process that can evaluate and approve agentic deployments — the other pillars operate in an organizational vacuum. Agents get deployed without adequate review, incidents

occur without clear ownership, and the program accumulates technical debt alongside governance debt in a combination that eventually requires painful remediation.

The governance pillar encompasses several distinct domains. Accountability assignment: who is responsible for each deployed agent, for the policy stack that governs it, and for the incident response when it fails? In most organizations, this assignment does not exist at the start of an agentic program and must be created, typically through a governance charter and the establishment of an AI program office or center of excellence. Policy development: the acceptable-use policy for agents, the model risk policy that governs model selection and evaluation, the data classification policy that determines what data agents can access, and the incident response process specific to agent failures. Regulatory compliance: the mapping of deployed agents to applicable regulatory requirements (EU AI Act risk classification, sectoral rules, state-level requirements) and the maintenance of the documentation and audit trail those requirements demand.

Governance failures are the most common cause of enterprise agentic program setbacks that are not immediately visible in technical metrics. An agent that is technically well-built but deployed without adequate governance review may perform well for months before a compliance gap, a policy violation, or an accountability question surfaces in a context — regulatory inquiry, board audit, media incident — where the absence of governance infrastructure is deeply inconvenient. The organizations that avoid this experience build governance alongside capability, not after it.

Pillar II — Orchestration and Architecture

The orchestration pillar is where the technical substance of agentic AI lives: the frameworks, runtimes, evaluation systems, and guardrail infrastructure that make agents work reliably in production. It is the pillar most familiar to engineering teams — the one that maps most directly to the build-and-ship work that engineering organizations know how to do — and it is consequently often the most developed and the least often the limiting factor in a mature program. The more common pattern is that orchestration is developed while governance and integration lag.

Orchestration encompasses: the choice of agent framework (LangGraph, AutoGen, CrewAI, or proprietary platforms like ServiceNow AI Agent Fabric) and the implications of that choice for architecture, vendor dependency, and governance capability; memory architecture and the state management patterns that allow agents to maintain context across sessions and tasks; evaluation infrastructure — the harness of automated tests, human review processes, and metric dashboards that provide the continuous quality signal needed to detect capability drift and policy violations; and guardrails, the runtime policy enforcement that prevents agents from taking actions outside their defined permission envelope.

The orchestration pillar is also where observability lives. An orchestration stack without adequate observability is a production deployment without instruments — you will discover problems from their effects rather than from their early signals, and the effects of agentic failures

are often more expensive than the failures themselves. Trace-level, metric-level, and evaluation-level observability should be designed into the orchestration architecture from the start, not bolted on after the first incident.

Pillar III — Use Case Identification

The use case pillar is about selection: choosing the right work for agents to do. This sounds like a strategy question, and it partly is, but it is also an engineering and governance question. Not all tasks are equally well-matched to agentic approaches; not all tasks that are well-matched carry the same risk profile; and not all tasks that look promising in a pilot environment scale successfully to production. The use case pillar is the structured process for making these distinctions and acting on them.

Use case identification involves a portfolio approach: mapping the landscape of candidate tasks, scoring them on value (what is the economic and strategic return if the agent succeeds?) and feasibility (how mature is the technology, how well-defined is the task, how reversible are the agent's actions?), and selecting a prioritized portfolio that balances quick wins (high feasibility, moderate value) with strategic investments (high value, higher complexity) and lighthouse projects (the use cases that demonstrate transformative capability and earn the next round of funding).

The human-in-the-loop design question — for each use case, at what points should a human review the agent's work, and what triggers escalation? — is also part of the use case pillar. This design question is not purely a safety question; it is also an economics question. Checkpoints that are too frequent recover the labor cost the agent was supposed to eliminate. Checkpoints that are too infrequent allow errors to propagate at scale. The right answer depends on the specific task, the agent's demonstrated quality at that task, and the organization's risk tolerance — and it should be revisited as the agent matures and quality metrics are established.

Pillar IV — Integration and Data

The integration pillar is, in many enterprise programs, the one that most consistently surprises. It is the pillar that determines whether an agent can actually do useful work, because an agent without reliable access to the systems and data it needs is, in practice, a capability demonstration rather than a production system. Integration work is also the pillar that most often hits organizational boundaries — the teams that own the CRM, the ERP, the data warehouse, and the identity provider may not be in the chain of command of the team building the agent, and their cooperation is not guaranteed.

Integration encompasses: system connectivity, the technical work of connecting the agent to enterprise systems through APIs, MCP servers, or direct database access; identity and access management, the design and implementation of the agent's identity model, its credential management, and its scoped access to enterprise systems; data readiness, the quality, currency, and governance of the data that the agent will retrieve and act upon; and interoperability

protocols, the adoption of standards like MCP and A2A that allow agents to communicate with tools and with each other in a governed, auditable way.

Data readiness is frequently the binding constraint in integration. Organizations discover, through the process of building agentic systems, that their data is less clean, less well-governed, and less accessible than they believed. Knowledge bases contain stale documents. CRM data has quality problems that were acceptable when humans were reading it but are unacceptable when an agent is acting on it at scale. Data classification is inadequate to support the access control requirements of an agent that operates across many domains and serves users with different permission levels. Addressing these problems takes time and organizational will — and the time to start is before the agent deployment rather than after the pilot has demonstrated that the data is a problem.

The Pillar Interactions

The four pillars are not independent. Each depends on the others, and weakness in one propagates. A strong orchestration stack without adequate governance is a capable but ungoverned agent program — technically impressive, organizationally vulnerable. Strong governance and orchestration without adequate integration produces agents that are well-governed and well-built but cannot access the systems needed to be useful. Excellent use case selection without the integration and orchestration to support the selected use cases produces a roadmap that looks compelling and lands nowhere.

The most effective approach to building agentic readiness is to develop all four pillars in parallel, with a shared timeline and explicit dependencies between them. The governance pillar should set the policy requirements that the orchestration pillar implements as guardrails. The use case pillar should be informed by the integration pillar's realistic assessment of what is achievable with the data and systems available. The orchestration pillar's evaluation infrastructure should be informed by the specific failure modes that the governance pillar's risk analysis has identified as most important to detect and measure.

This parallel development requires cross-functional coordination — between the engineering team building the orchestration stack, the governance team developing policies, the business teams identifying use cases, and the platform and data engineering teams managing integration. That coordination is itself a governance challenge, and it is the one that most commonly falls through the cracks in organizations that are accustomed to sequencing technology and governance as separate workstreams. The agentic enterprise is an integrated system; it requires integrated development.

> *"A program that stands on three pillars and ignores the fourth will tip over within a year. The tipping usually happens in a way that was entirely predictable from the structure of the program — and entirely surprising to the leadership who approved it."*

PROGRAM DESIGN NOTE

When establishing an enterprise agentic AI program, map your initial budget and headcount allocation across the four pillars explicitly. If more than 60% of the initial investment is in orchestration and use cases (the technically interesting work) and less than 40% is in governance and integration (the work that enables everything else), the allocation is likely to produce a program that creates impressive demos and struggles to reach production at scale. Rebalancing toward governance and integration early is cheaper than doing so after a production incident makes the imbalance undeniable.

PART II

The Four Pillars of Readiness

Sixteen chapters — four per pillar — laying out governance, orchestration, use case identification, and integration in the depth a CIO needs.

CHAPTER 13

The Governance Charter

Pillar I — who owns the agents, who answers when they fail

> Every serious agentic program eventually reaches the same inflection point: the agent does something unexpected, the humans look at each other, and no one is entirely sure who owns the problem. A governance charter exists precisely to answer that question before the moment arrives. It names the individuals and bodies with authority over an agent program, defines the thresholds at which decisions escalate, and creates the institutional memory that outlasts any single deployment cycle. Without one, accountability diffuses into a fog of shared responsibility that protects no one and disciplines nothing.

Why Agents Break Normal IT Governance

Traditional IT governance was built for deterministic systems. A database either returns the row or it doesn't. A workflow either fires the trigger or it times out. The audit trail for a traditional system is, in principle, complete: every state change has a cause, and every cause can be traced to an instruction a human wrote. Agentic systems violate this assumption at the architectural level. The model reasons; the model plans; the model chooses among tools it has never explicitly been told to use in that combination. The result is a class of decisions that are emergent rather than authored, and conventional governance frameworks have almost no vocabulary for emergent decisions.

This matters practically because most enterprise risk committees operate on the assumption that software does what it is programmed to do. When an agent interprets an ambiguous instruction more liberally than its operators intended — forwarding a confidential attachment, approving a purchase order, deleting a file — the gap between "what we deployed" and "what it did" becomes the central governance problem. The charter must create structures that survive that gap.

Anatomy of a Governance Charter

A governance charter for an agentic program typically contains five interlocking elements. First, a **scope statement** that identifies which agents and agent classes fall under the charter's jurisdiction — distinguishing, for instance, between a narrow task-execution agent that calls a single internal API and a multi-step research agent with broad web access and the ability to send email on behalf of a named employee. Second, an **ownership map** that identifies a named business owner (accountable for outcomes), a technical owner (accountable for design and

operation), and a data owner (accountable for the information the agent can access). These three roles must be distinct people.

Third, the charter defines a **decision authority matrix** — a table that maps classes of agent action to the level of human approval required. Actions that are reversible, low-value, and bounded (retrieving a document, drafting an email) sit in one tier; actions that are irreversible, high-value, or boundary-crossing (executing a payment, modifying a production database, communicating externally on behalf of an executive) sit in another. The matrix is the governance charter's most operationally consequential section, because it defines where the agent is permitted to act autonomously and where it must pause for a human.

Fourth, an **incident and escalation protocol** that names specific response roles, defines what constitutes an "agentic incident" (broadly: any unintended action with external or irreversible consequences), and prescribes timelines for notification, containment, and post-incident review. Fifth, a **review cadence** — quarterly at minimum for high-autonomy agents — at which the charter itself is revisited in light of new capabilities, new risks, and operational experience.

> *"Governance is not a tax on innovation. It is the institutional form of institutional memory. The organization that cannot say who owns the agent cannot learn from what the agent does."*

The Three Ownership Questions

When an agent causes harm — an incorrect action, a data exposure, a regulatory breach — three ownership questions arise in rapid succession. Who decided to deploy this agent? That is the business owner's domain. Who decided how it was built and constrained? That is the technical owner's domain. Whose data did it use and who was responsible for its classification and access controls? That is the data owner's domain. Most organizations conflate two or all three of these roles in a single person, usually a product manager or an ML engineer, who is then left holding a problem that requires authority they were never granted.

A charter that separates these three ownership lines creates the organizational surface area for genuine accountability. The business owner signs off on the risk appetite embedded in the decision authority matrix. The technical owner signs off on the agent's architecture, its tool permissions, and its fallback behaviors. The data owner signs off on the classification labels that determine what the agent can and cannot see. All three must agree before a new agent class is deployed, and all three are notified within a prescribed window when an incident occurs.

The Oversight Body

For organizations running more than a handful of agents, a single charter per deployment quickly becomes unmanageable. The natural evolution is an **AI Governance Council** — a cross-functional body that owns a portfolio of charters rather than individual ones, sets enterprise-wide standards for agent classification, and maintains the institutional relationship with legal,

compliance, and audit. Gartner's Trust, Risk, and Security Management (TRiSM) framework offers a useful organizing lens here: the Council's mandate spans trustworthiness, reliability, and privacy — not just security.

The Council should include representatives from legal, information security, data governance, the business units deploying agents, and HR (for any agent that affects employees). A CISO or Chief AI Officer, where one exists, typically chairs it. The Council's output is not policies but decisions: approvals, escalations, and periodic re-assessments of agents whose risk profile has shifted. It should meet at least monthly, with standing authority to convene an emergency session when an agentic incident exceeds predefined severity thresholds.

From Charter to Culture

A governance charter filed in a SharePoint folder and never consulted is not governance. The charter becomes operational only when it is embedded in the deployment workflow — specifically, when no agent can be promoted to a production environment without a signed charter that names its three owners, its decision authority tier, and its incident escalation path. That embedding requires tooling: an internal registry that tracks every deployed agent, its charter status, its current review date, and any open incidents against it.

Over time, the charter process itself becomes a cultural signal. Teams that design their agents to require minimal decision authority — because they have thought carefully about reversibility and scope — are building differently from teams that design first and governance-audit later. The former approach produces systems that are genuinely safer because their architects internalized the ownership questions before the first line of code was written.

> CHARTER MINIMUM VIABLE CHECKLIST
>
> A governance charter is not a tome. The minimum viable version fits on two pages: scope and agent class, three named owners with contact details, a decision authority table with at least three tiers, an incident definition and 24-hour notification requirement, a quarterly review date, and signatures from all three owners. If you cannot produce this document, you do not have a charter — you have an intention.

CHAPTER 14

The Policy Stack

Acceptable use, model risk, data classification, third-party AI, incident response

5
policy domains in a complete stack

SR 11-7
Federal Reserve model risk guidance, adapted for agents

> A governance charter names who is responsible. A policy stack defines what is allowed. The distinction matters because the same agent can be deployed in dozens of different contexts across a large enterprise, and each context carries its own risk profile, its own regulatory obligations, and its own tolerance for autonomous action. The policy stack is the body of living documents that governs all of those contexts simultaneously — not by anticipating every scenario, but by establishing principles and decision rules that teams can apply when novel situations arise. Getting the stack right is less a legal exercise than an operational engineering problem.

Acceptable Use Policy

The foundation of the stack is an **acceptable use policy** (AUP) for agentic systems. It differs from a conventional AUP in two important ways. First, it must address the actions of the agent as well as the actions of the user: a policy that only constrains what humans type into a chat interface provides no guidance on what the agent is permitted to do with the tools it has been given. Second, it must address probabilistic outputs: unlike a database query that returns a definitive result, an agent's output is a best-effort inference, and the AUP must establish how that uncertainty should be disclosed to downstream consumers of the agent's work.

A well-drafted AUP specifies approved use cases (the classes of task for which the agent is sanctioned), prohibited use cases (tasks that are explicitly off-limits regardless of apparent capability), handling rules for sensitive data categories encountered unexpectedly, and disclosure requirements when an agent's output is incorporated into a communication, a decision, or a deliverable that reaches an external party. The AUP should be reviewed annually and whenever a significant new capability — a new tool integration, a new model version, or a new deployment context — is introduced.

Model Risk Management

Financial services regulators have required **model risk management** (MRM) for algorithmic decision systems since the Federal Reserve's SR 11-7 guidance in 2011. That framework — model

inventory, validation, monitoring, and governance — applies to agentic AI with some important extensions. The model in an agentic system is often a third-party foundation model that the organization did not train and cannot fully audit. MRM for agents therefore requires a vendor risk component that tracks the model provider's change management practices, their evaluation methodology, and their contractual commitments around performance stability.

The validation challenge is particularly acute. Traditional MRM validates a model on a held-out test set and declares it fit for purpose within a defined input distribution. An agent operates across a far wider and less predictable input distribution — including adversarially crafted inputs from external actors — and its outputs include not just predictions but actions with real-world consequences. Model risk teams adapting to this environment are increasingly building **behavioral test suites**: structured collections of scenarios designed to probe the agent's behavior at the edges of its approved use cases, including scenarios that involve conflicting instructions, ambiguous data, and attempts at prompt injection.

Data Classification for Agents

Most enterprise data classification schemes were designed for static documents and database records. They answer the question: how sensitive is this data at rest? For agents, the more important question is: how sensitive is this data in motion through an inference pipeline? An agent that retrieves a customer record (classified as Confidential), summarizes it (output now contains Confidential data), and sends the summary to an external vendor has moved data across a classification boundary in a way that no DLP rule written for email or file transfer will catch.

The policy response is to extend classification labels downstream: the output of any operation on Confidential data inherits a Confidential label unless an explicit declassification rule applies. This sounds straightforward but requires tooling — specifically, a way to attach classification metadata to the agent's working context and propagate it through tool calls, so that the agent's output-handling logic can check whether it is permitted to pass that output to its next destination. Several enterprises are implementing this as a **context tagging system**: a lightweight key-value store attached to the agent's session that accumulates classification labels as the agent retrieves data and is consulted before any outbound action.

> *"The data classification problem for agents is not a new problem; it is an old problem running at a speed that existing controls were not designed to match."*

Third-Party AI Policy

Most agentic deployments rely on third-party components at multiple layers: a foundation model from a hyperscaler or frontier lab, an orchestration framework from an open-source project or commercial vendor, and various tool integrations that reach into external services. Each of these

introduces a supply-chain risk that a third-party AI policy must address. The policy should require a minimum security baseline for any AI component integrated into a production agent — covering at minimum: data processing agreements, incident notification timelines, model change notification, and access to evaluation documentation sufficient to meet the organization's MRM requirements.

The emergence of the Model Context Protocol and similar standards is changing the third-party AI landscape: tool integrations that once required bespoke development can now be sourced from a growing ecosystem of pre-built MCP servers. This lowers integration cost dramatically, but it also means that the policy must address a new class of third-party component — the MCP server — whose security properties and data handling practices vary enormously. A third-party AI policy that was written before MCP existed will almost certainly need to be updated.

Incident Response for Agents

The incident response plan for an agentic system shares the general structure of a cybersecurity incident response plan — detect, contain, eradicate, recover, post-incident review — but requires agent-specific playbooks for the failure modes that are unique to autonomous systems. Three of these are particularly important. First, the **runaway loop**: an agent enters a state where it repeatedly calls a tool or produces a sequence of actions that is not converging. The playbook must include a mechanism for hard-stopping the agent without losing the state needed for post-incident analysis. Second, the **unauthorized data exfiltration**: an agent passes data to an unintended destination, either through a misconfigured tool or through a prompt injection attack. The playbook must specify who is notified, in what timeframe, and what forensic preservation steps are required. Third, the **model substitution**: a model provider silently updates the underlying model, and the agent's behavior changes in ways that violate its decision authority matrix. The playbook must specify how behavioral regression testing is triggered and what the threshold is for rolling back to a prior model version.

> POLICY DEBT ACCUMULATES FAST
>
> Organizations that deploy agents before writing policy are not saving time — they are borrowing it at high interest. Every production agent deployed without an acceptable use policy, a data classification extension, and an incident playbook is a governance liability that will eventually need to be retrofitted. Retrofitting policy onto a deployed system is harder than writing policy before deployment, because the system has already established operational patterns that the policy must either ratify or require to be changed.

CHAPTER 15

Walking the NIST AI RMF

Govern, Map, Measure, Manage — applied to a real agent program

4
NIST RMF functions applied to agentic programs

40%
of agentic risk scenarios not captured by standard taxonomy (CSA pilot)

Q4 2026
target for NIST AI Agent Interoperability Profile

> The NIST AI Risk Management Framework was released in January 2023 as a voluntary, non-prescriptive guide for managing the risks of AI systems across their entire lifecycle. Its four functions — Govern, Map, Measure, Manage — form a practical loop that organizations can walk regardless of their sector, their regulatory context, or the specific AI technology they are deploying. Walking the RMF for an agentic program is different from walking it for a conventional predictive model, because the risk surface is wider, the failure modes are more diverse, and the velocity of change is higher. This chapter follows that walk concretely, pausing at each function to identify where the standard guidance strains and where enterprise practitioners have had to improvise.

Govern: The Foundation

The Govern function is about creating the organizational conditions under which responsible AI risk management can happen. In the RMF's framing, Govern asks: does the organization have policies, roles, and processes in place to make risk-informed decisions about AI? For an agentic program, the Govern function maps directly to the governance charter and policy stack discussed in the preceding two chapters. But the RMF adds a specific emphasis that practitioners sometimes underweight: **accountability structures must be documented and tested, not merely declared**.

Testing accountability means running tabletop exercises that simulate agentic incidents and asking: who actually gets called, in what order, and do they have the authority and the information they need to respond effectively? Organizations that skip this step frequently discover, during an actual incident, that their accountability structures are aspirational rather than operational — the CISO is listed as the incident commander but has no runbook for an agent that has exfiltrated data through a tool the security team didn't know existed.

The RMF's Govern function also addresses organizational culture. The framework is explicit that risk management cannot succeed if it is treated as a compliance exercise rather than a genuine operational priority. For agentic AI, this cultural dimension is particularly important because the technology is moving faster than any governance framework can track, and the only durable

protection is a team that is genuinely curious about failure modes rather than one that is merely checking boxes on an audit schedule.

Map: Understanding the Risk Surface

The Map function asks organizations to identify and categorize the risks associated with a specific AI system in a specific deployment context. For conventional models, this typically means a risk taxonomy organized around model performance dimensions: accuracy, fairness, robustness, privacy. For agents, the Map function must extend to cover a substantially wider risk surface that includes the agent's tools, its memory architecture, its multi-agent interactions, and the potential for adversarial manipulation of any of these components.

The concept of an **"agentic fitness gap"** has emerged from practitioners attempting to apply the standard RMF risk taxonomy to agentic systems. The gap refers to the distance between what the RMF's default risk categories capture and what actually goes wrong with agents in production. Standard categories like "bias" and "accuracy" are meaningful for agentic systems, but they miss the failure modes that are most operationally significant: unauthorized tool use, context window poisoning, cross-agent trust exploitation, and resource exhaustion from runaway loops. Closing the fitness gap requires extending the Map function's risk taxonomy with agent-specific categories.

The Cloud Security Alliance has proposed an Agentic Profile extension to the NIST RMF that adds exactly these categories, along with a structured process for mapping the trust relationships between agents in a multi-agent system. The extension remains in draft form as of early 2026, but several large financial services organizations have begun piloting it in their internal RMF implementations.

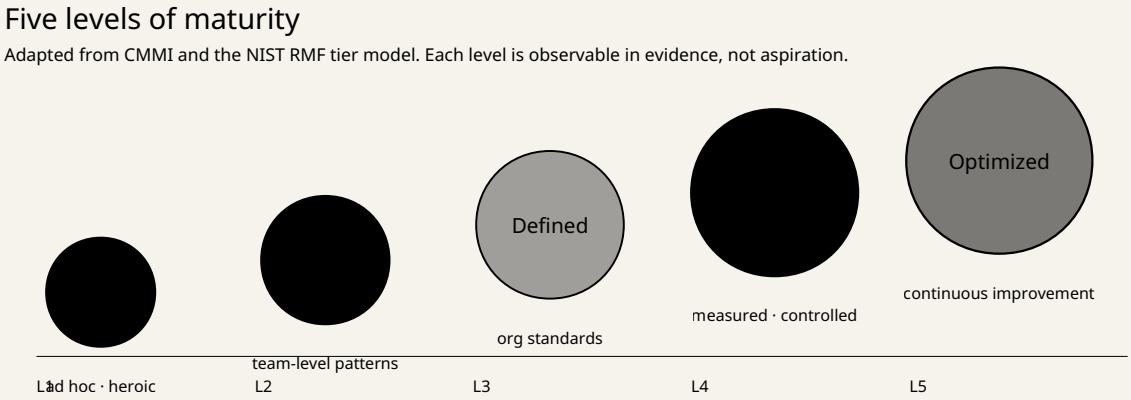

FIGURE 15.1FIVE LEVELS. THE JUMP FROM L2 TO L3 IS WHERE MOST AGENT PROGRAMS STALL — IT REQUIRES STANDARDS, NOT JUST TEAMS.

Measure: Evaluating What You Cannot Fully Observe

The Measure function asks organizations to assess, analyze, and track the risks they have identified. For conventional models, measurement typically means evaluating performance metrics on a held-out test set and tracking those metrics in production through model monitoring. For agents, measurement is fundamentally harder because the agent's behavior in production is not a simple mapping from inputs to outputs — it is a sequence of decisions, tool calls, and state transitions, many of which are not directly observable through conventional monitoring.

The NIST AI Agent Standards Initiative, announced in February 2026, is developing an AI Agent Interoperability Profile planned for Q4 2026 that will, among other things, define standard telemetry formats for agentic systems. Until that profile is available, organizations are improvising: using OpenTelemetry traces to capture the sequence of tool calls and model invocations that constitute an agent run, and building custom evaluation harnesses that measure behavioral properties — instruction-following fidelity, tool use appropriateness, escalation correctness — rather than just output quality.

The Measure function also includes an often-overlooked component: **uncertainty quantification**. An agent that is uncertain about the right action is a different risk profile from an agent that is confidently wrong. Emerging evaluation frameworks like Langfuse and Arize Phoenix are beginning to expose uncertainty signals from the underlying model as part of their trace data, which allows risk teams to flag runs where the agent acted confidently in a domain where its historical accuracy has been low.

> *"The NIST RMF's Manage function is where governance meets engineering. An organization that has governed and mapped and measured but cannot stop a running agent that is behaving badly has built a risk program that ends at the page."*

Manage: Operationalizing the Response

The Manage function asks organizations to prioritize and respond to the risks they have measured. In the RMF's framing, this includes both proactive risk treatment — changing the system to reduce risk before it materializes — and reactive incident response — containing and remediating harms after they occur. For agents, the Manage function maps to three concrete operational capabilities: the ability to modify an agent's tool permissions without redeploying it, the ability to roll back to a prior model version when a behavioral regression is detected, and the ability to hard-stop a running agent without losing the state needed for post-incident analysis.

These three capabilities are not difficult to build, but they require intentional design. An orchestration architecture that cannot modify tool permissions at runtime — because they are baked into the agent's system prompt rather than managed through a policy engine — cannot execute the Manage function's risk treatment actions quickly enough to prevent a slow-moving

harm from becoming a large one. The alignment between the RMF's Manage function requirements and the architectural choices in the orchestration stack is one of the most important and least-discussed connections in enterprise agentic AI deployment.

> THE AGENTIC FITNESS GAP IN PRACTICE
>
> A major European bank piloting the CSA Agentic Profile extension found that roughly 40% of the risk scenarios their MAP exercises surfaced were not captured by the standard NIST RMF risk taxonomy. The missing scenarios fell into three clusters: trust exploitation between agents in a multi-agent pipeline, context window manipulation through adversarially crafted tool responses, and resource exhaustion through recursive planning loops. None of these were novel risks — they had been identified by security researchers — but they had not been systematically incorporated into the bank's existing AI risk management practice.

CHAPTER 16

ISO/IEC 42001 and the EU AI Act

Certification, conformity assessment, and the audit trail of an autonomous system

8
Annex A control domains in ISO/IEC 42001

Articles 8–15
EU AI Act high-risk obligations

> Two regulatory instruments now define the outer walls of enterprise AI governance for organizations operating in Europe or with European customers. ISO/IEC 42001, published in December 2023, is the first certifiable management system standard for AI — a PDCA-cycle framework that gives organizations a structured, auditable path to responsible AI management. The EU AI Act, which entered into force in August 2024, is binding law with graduated obligations based on risk classification. Together they create a compliance architecture that is more demanding than anything the enterprise software industry has previously encountered, and for agentic systems specifically, that architecture contains several provisions that require careful engineering, not just careful documentation.

ISO/IEC 42001: The PDCA Backbone

ISO/IEC 42001 is structured like other ISO management system standards — ISO 9001 for quality, ISO 27001 for information security — around the Plan-Do-Check-Act cycle. An organization that implements 42001 establishes an AI management system: a documented set of policies, processes, roles, and records that demonstrates it is managing AI risks in a systematic and continuously improving way. The standard is certifiable, meaning a third-party audit body can assess conformity and issue a certificate — a capability that is becoming increasingly important as enterprise customers and regulators begin requiring documented AI governance from their vendors and counterparties.

Annex A of the standard contains a library of controls organized around eight domains: AI system impact assessment, AI system life cycle, data management, technical controls, documented information, supply chain, competence and awareness, and stakeholder engagement. For agentic systems, the most operationally demanding controls sit in the technical domain: specifically, the requirements for **explainability**, **robustness testing**, and **human oversight mechanisms**. An agent that cannot provide a human-interpretable account of why it took a specific action at a specific moment will fail the explainability control; an agent that has not been tested against adversarial inputs will fail the robustness testing control; an agent deployed without a documented mechanism for human override will fail the human oversight control.

The standard also requires an **AI impact assessment** process — analogous to the privacy impact assessments required under GDPR — that evaluates the potential impacts of an AI system on individuals, organizations, and society before deployment and at regular intervals thereafter. For high-autonomy agents with the ability to affect significant resources or make consequential decisions, this assessment is a substantial undertaking that requires input from legal, ethics, and subject matter experts who understand the deployment context.

EU AI Act: Obligations for High-Risk Systems

The EU AI Act classifies AI systems into four risk tiers: unacceptable risk (prohibited), high risk (subject to conformity assessment and ongoing obligations), limited risk (transparency requirements), and minimal risk (no specific obligations). Most agentic systems deployed in consequential enterprise contexts — those affecting employment decisions, credit assessments, safety-critical operations, or public services — are likely to qualify as high-risk under Annex III of the Act. High-risk classification triggers a comprehensive set of obligations under Articles 8 through 15 that collectively define one of the most demanding technical compliance regimes in the history of software regulation.

Article 9 requires a risk management system that is continuously updated throughout the system's lifecycle — not a one-time risk assessment but an ongoing process that tracks identified risks, their likelihood and severity, and the measures taken to mitigate them. **Article 10** requires that training, validation, and testing data meet defined quality standards — a requirement that applies not just to the foundation model but to any fine-tuning, retrieval corpus, or tool response data that the agent uses at runtime. **Article 11** requires technical documentation that is comprehensive enough to allow a notified body to assess conformity — including documentation of the system's intended purpose, its performance characteristics, its limitations, and its foreseeable misuse scenarios.

Article 12 requires automatic logging of events throughout the system's operation — with specific requirements for the logging of operations that are "relevant for the identification of risks." For agents, this means logging not just the final output but the intermediate steps: the tool calls made, the data retrieved, the reasoning produced at each planning step. **Article 13** requires transparency toward users that is sufficient for them to interpret the system's output correctly, including disclosure that they are interacting with an AI system. **Article 14** requires human oversight mechanisms that are effective — meaning the oversight must be technically capable of preventing or minimizing risks, not merely a paper control. **Article 15** requires that the system achieves an appropriate level of accuracy, robustness, and cybersecurity, with specific attention to resilience against attempts to alter outputs through adversarial manipulation.

> *"The EU AI Act's Article 14 requirement for effective human oversight is the one provision that most directly constrains the autonomy architecture of a deployed agent. 'Effective' means technically capable of intervention, not merely nominally available."*

Conformity Assessment and the Audit Trail

For most high-risk AI systems, the EU AI Act requires a conformity assessment before the system can be placed on the market. For some categories — biometric identification, critical infrastructure, employment — this assessment must be conducted by a **notified body**, a third-party organization accredited by a national authority. For other high-risk categories, self-assessment is permitted, but the organization must produce a **Declaration of Conformity** and register the system in the EU AI Act's public database before deployment.

The audit trail that supports conformity assessment is substantially more demanding than the audit trail required for conventional software compliance. It must include: the technical documentation required under Article 11; records of the quality management system required under Article 17; records of the post-market monitoring system required under Article 72; and, for any serious incidents, the incident report required under Article 73. For agents specifically, the requirement to log "operations relevant for the identification of risks" under Article 12 effectively mandates complete execution traces — not summarized logs but the full sequence of model calls, tool invocations, and state transitions that constitute each agent run.

The FRIA Frontier

The EU AI Act introduces a new instrument that has no direct analog in prior EU technology regulation: the **Fundamental Rights Impact Assessment** (FRIA). Required for high-risk AI systems deployed by public bodies and certain private operators in high-stakes contexts, the FRIA extends the standard impact assessment framework to explicitly consider the impacts of the AI system on fundamental rights — the rights protected under the EU Charter of Fundamental Rights. For agents operating in contexts that affect access to essential services, employment, education, or justice, the FRIA is a significant undertaking.

Conducting a FRIA for an agentic system requires engaging with questions that are genuinely novel: how does the agent's behavior affect the rights of the people it interacts with, and how does that effect change when the agent is operating at the scale and speed that autonomous systems make possible? An agent that processes hundreds of loan applications per hour, making decisions that affect people's access to credit, is not simply a faster version of a human loan officer; it is a qualitatively different kind of decision-maker whose systematic biases can propagate at a speed and scale that makes them far harder to detect and correct. The FRIA is designed to surface exactly these systemic effects before deployment, rather than after they have caused harm at scale.

42001 VS. THE EU AI ACT: COMPLEMENTARY, NOT INTERCHANGEABLE

ISO/IEC 42001 certification does not constitute conformity with the EU AI Act, and EU AI Act compliance does not substitute for 42001 certification. The two instruments operate at different levels: 42001 is a management system standard that demonstrates organizational capability; the EU AI Act is binding law that mandates specific technical and documentation requirements for specific system types. Organizations serving European markets will ultimately need both — and the overlap between the two frameworks is substantial enough that implementing them in an integrated way is considerably more efficient than treating them as separate compliance workstreams.

CHAPTER 17

The Orchestration Stack

Pillar II — frameworks, runtimes, and the plumbing of multi-step work

> If governance is the organizational layer that defines who owns the agent and what it is permitted to do, orchestration is the technical layer that actually makes it do it. An orchestration stack is the collection of frameworks, runtimes, and plumbing components that translate a goal into a sequence of model calls, tool invocations, and state updates — and that manages the coordination between multiple agents when the task requires more than one. The choice of orchestration architecture is not merely a developer preference; it is a risk architecture decision that determines what kinds of failures the system is capable of, what kinds of interventions are possible at runtime, and how well the system's behavior can be observed and audited.

The Orchestration Layer Defined

The orchestration layer sits between the application that initiates a task and the models and tools that execute it. Its responsibilities include: decomposing the task into subtasks, sequencing those subtasks according to their dependencies, routing each subtask to the appropriate model or tool, managing the shared context that flows between steps, handling errors and retries at the step level, and surfacing the information needed for human review at appropriate checkpoints. In a single-agent system, orchestration is relatively straightforward — essentially a loop over the agent's plan-act-observe-remember cycle. In a multi-agent system, orchestration becomes substantially more complex because it must also manage the communication protocols, trust relationships, and state synchronization between agents that may be running on different runtimes with different capabilities and different permission sets.

The orchestration layer is where most of the interesting engineering decisions in an agentic deployment are made, and it is where most of the security controls that matter at runtime are implemented. An orchestration framework that provides no hooks for runtime policy enforcement — no way to inspect a tool call before it executes, no way to modify the agent's context based on a policy decision, no way to pause execution for human review — cannot implement the governance charter's decision authority matrix in practice, regardless of how carefully that matrix has been designed.

LangGraph: Directed Graphs for Complex Workflows

LangGraph represents the current state of the art in explicit, graph-based orchestration. Rather than expressing agent behavior as a monolithic loop, LangGraph asks the developer to model it

as a directed graph of nodes and edges, where nodes are computation steps (model calls, tool calls, conditional branches) and edges are the transitions between them. This representation makes the agent's possible execution paths explicit and statically analyzable — a significant advantage from a governance perspective, because it means the set of things the agent can do is defined by the graph structure, not emergent from the model's reasoning.

LangGraph's stateful execution model — where each node in the graph can read and write a typed state object that persists across the entire run — makes it well-suited for complex, multi-step workflows that require careful state management. The framework's support for human-in-the-loop checkpoints, where execution pauses and a human must approve or modify the state before the graph continues, is particularly valuable for agents that operate in high-stakes domains where the governance charter requires human oversight at specific decision points. LangGraph's **persistence layer** can checkpoint the full graph state to an external store, enabling long-running agents that can be interrupted and resumed and providing the forensic trail needed for post-incident analysis.

AutoGen and CrewAI: The Multi-Agent Dimension

AutoGen, Microsoft's multi-agent orchestration framework, takes a different approach: it models agents as conversational participants in a group chat, where agents exchange messages and the orchestrator routes those messages according to a defined policy. This conversational model makes it easy to build systems where the decomposition of a task emerges from the agents' interaction rather than being specified in advance — useful for exploratory tasks where the right decomposition is not known at design time, but harder to audit because the agent's behavior is less constrained by the framework's structure.

CrewAI offers a higher-level abstraction: agents are defined as roles (Researcher, Writer, Reviewer) with associated tools and goals, and tasks are assigned to roles rather than to specific model instances. The framework handles the coordination — including parallel execution of independent tasks and sequential execution of dependent ones — and provides a structured output format that makes it easier to integrate agent work products into downstream systems. CrewAI's role-based model maps naturally onto organizational structures, which makes it attractive for enterprise teams that want to reason about their agent deployments in familiar terms.

The choice between LangGraph's explicit graph model, AutoGen's conversational model, and CrewAI's role-based model is ultimately a choice between different tradeoffs on the spectrum from control to flexibility. Organizations with strong governance requirements and well-defined workflows tend toward LangGraph; organizations building exploratory or research agents where the workflow is inherently open-ended tend toward AutoGen; organizations that want to align their agent architecture with their organizational structure tend toward CrewAI.

> *"The orchestration framework you choose is not a neutral technical decision. It determines what you can observe, what you can control, and what you can audit. Choose it with the same deliberateness you would apply to selecting a database architecture."*

Runtimes and Plumbing

Beneath the orchestration framework sit the runtime components that actually execute the agent's actions: the inference API that calls the model, the tool execution environment that runs the functions the agent invokes, and the message broker that routes information between agents in a multi-agent system. Each of these has implications for the security and observability of the overall system that practitioners frequently underestimate when they are focused on the higher-level orchestration logic.

The **tool execution environment** deserves particular attention. A tool that the agent can call is, by definition, code that executes with some level of privilege in the organization's environment. The principle of least privilege — already well-established in conventional software security — applies to agent tools with additional force, because the agent may call tools in combinations and sequences that the tool developers did not anticipate. Tool execution should be sandboxed wherever possible, with explicit grants of the permissions each tool requires and automatic revocation when the agent's session ends. The AWS IAM model of fine-grained, time-limited, scope-specific permissions is a useful template for thinking about tool permission architecture.

The Orchestration Stack as Control Surface

A mature orchestration architecture treats the stack not just as a mechanism for executing tasks but as the primary control surface for runtime risk management. This means instrumenting every layer: traces at the model call level that capture the prompt, the response, the tool calls, and the latency; metrics at the workflow level that track success rates, escalation rates, and cost per run; alerts at the policy level that fire when the agent's behavior approaches or crosses a defined threshold. The orchestration stack that provides this instrumentation out of the box is not just more observable — it is more governable, because the governance charter's escalation thresholds can be implemented as runtime controls rather than as post-hoc audits.

> FRAMEWORK SELECTION CHECKLIST
>
> Before committing to an orchestration framework for a production agent, answer four questions: Does it support human-in-the-loop checkpoints at arbitrary graph nodes? Does it provide a persistence layer for checkpointing and recovery? Does it expose traces in a format compatible with your observability stack? Does it support runtime modification of tool permissions without redeployment? A framework that cannot answer yes to all four imposes architectural debt that will be expensive to retire.

CHAPTER 18

Memory and State

Short-term, long-term, episodic — what an agent needs to remember to be useful

3

primary memory types: in-context, external, parametric

> Memory is the property that distinguishes a useful agent from a stateless question-answering system. Without memory, every interaction starts from the same blank slate, every piece of context must be re-supplied by the user, and the agent cannot accumulate the understanding of a domain, a user, or an ongoing task that makes sophisticated autonomous work possible. But memory also introduces some of the most significant risks in agentic deployment: an agent that remembers incorrectly is more dangerous than one that forgets, because it acts with unwarranted confidence; an agent that can be made to remember adversarially crafted content is a security vulnerability; and an agent whose memory is accessible to unauthorized parties is a data governance failure. Getting memory right requires thinking carefully about what the agent needs to remember, for how long, in what form, and under what access controls.

Three Kinds of Memory

Agentic memory is typically categorized into three types that differ in their scope, duration, and storage mechanism. **In-context memory** is the simplest: it is the content of the model's context window during a single agent run. Everything the agent knows during a run — the system prompt, the conversation history, the tool results, the intermediate reasoning — lives in the context window. In-context memory is fast and flexible, but it is bounded by the context window size (which, even at 128K or 200K tokens, is finite) and it evaporates entirely when the run ends. It is appropriate for single-turn tasks and short workflows, but insufficient for anything that spans multiple sessions or requires knowledge accumulated over time.

External memory extends the agent's effective knowledge by connecting it to a retrieval system: a vector database, a document store, a structured knowledge graph, or some combination of these. When the agent needs information that exceeds the context window, it issues a retrieval query and incorporates the results into its working context. External memory can be vast — terabytes of documents, years of interaction history — and it persists across runs. But retrieval introduces a new failure mode: the agent may retrieve information that is outdated, irrelevant, or adversarially crafted, and it has no reliable way to distinguish good retrieval results from bad ones without additional verification mechanisms.

Parametric memory is knowledge encoded in the model's weights through training and fine-tuning. Unlike in-context and external memory, parametric memory cannot be updated at runtime — changing it requires retraining or fine-tuning the model, which is expensive and time-consuming. Parametric memory is valuable for stable, domain-specific knowledge that the agent will need constantly — the vocabulary of a specialized field, the conventions of a particular document type, the behavioral policies that should always be active — but it is the wrong mechanism for knowledge that changes frequently or that must be auditable.

Episodic Memory and the Accumulation of Experience

A fourth category — **episodic memory** — is increasingly recognized as a distinct and important type. Episodic memory stores structured records of past agent runs: what task was attempted, what steps were taken, what succeeded and what failed, and what the outcome was. Unlike external memory, which stores factual content, episodic memory stores procedural history — the agent's accumulated experience of how to approach classes of task. An agent with access to its episodic memory can recognize that a new task is similar to one it has successfully completed before and adapt its approach accordingly; it can also recognize that a proposed sequence of steps has historically led to failure and generate an alternative plan.

Building effective episodic memory requires solving several hard problems. The records must be structured consistently enough to be retrievable — which means designing a schema for episode representation that captures the relevant dimensions of variation across tasks. They must be indexed effectively — which means choosing retrieval mechanisms that can surface semantically relevant episodes, not just syntactically similar ones. And they must be maintained — which means building processes to review, curate, and when necessary correct the episodic store, because an agent that has learned the wrong lesson from a past failure is more dangerous than one that learned nothing.

> *"The agent that cannot learn from its own history is condemned to repeat it. But the agent that learns the wrong lessons from a poisoned history may be more dangerous than the amnesiac."*

State Management and the Persistence Problem

State management is the engineering problem of keeping track of where the agent is in a complex, multi-step workflow across interruptions, failures, and restarts. An agent that is interrupted mid-workflow — by a system timeout, a human-in-the-loop checkpoint, or an unexpected error — must be able to resume from where it left off without re-executing steps that have already been completed, without losing the context accumulated so far, and without violating the invariants of the workflow's state machine. This is a harder problem than it appears, because the state of an agent run is not just a set of values — it includes the model's internal state (which cannot be serialized directly), the pending tool calls (which may have side effects

that cannot be safely retried), and the conversation history (which may contain sensitive information that must be stored with appropriate access controls).

LangGraph's persistence layer addresses this problem through a checkpoint architecture: at each node in the execution graph, the full state is serialized and stored in an external store (PostgreSQL, Redis, or a cloud-native equivalent). If execution is interrupted, it can be resumed from the most recent checkpoint without re-executing prior steps. This checkpointing architecture also provides the foundation for the forensic trail required by the EU AI Act's Article 12 logging obligations: the checkpoint store is, in effect, a complete record of every state the agent passed through during its execution, which is exactly what an auditor needs to reconstruct the sequence of events that led to a particular outcome.

Memory Security and Data Governance

Memory stores are high-value targets for adversaries. An agent's vector store typically contains a condensed, semantically indexed representation of the organization's most important information — exactly the kind of information that an adversary wants to exfiltrate. Access controls on memory stores must be at least as rigorous as access controls on the source documents, and in practice they should be more rigorous, because the vector store's semantic indexing makes it easier to extract information efficiently than from the source documents themselves.

The specific attack known as **memory poisoning** — where an adversary injects crafted content into the agent's memory store with the intention of influencing its future behavior — is a significant and underappreciated threat. A prompt injection attack that succeeds in getting malicious instructions into the agent's episodic or external memory store is more dangerous than one that affects only a single run, because its effects persist across all subsequent runs that retrieve the poisoned content. Defense against memory poisoning requires both technical controls (input validation, anomaly detection on memory writes, human review of episodic memory updates) and architectural controls (separation between the agent's read path and write path, with stronger authentication required for writes).

> THE CONTEXT WINDOW IS NOT MEMORY
>
> A common misconception in early agentic deployments is that a large context window solves the memory problem. It doesn't. A 200K-token context window is large enough to hold a small book, but it is not large enough to hold the accumulated interaction history of a production agent running thousands of tasks per month. More importantly, stuffing the context window with everything the agent might possibly need is expensive (every token costs money), slow (inference latency scales with context length), and often counterproductive (models that receive more context than they need frequently exhibit degraded performance on the task at hand). Effective memory architecture selects what goes into the context window, it doesn't attempt to make the context window large enough to hold everything.

CHAPTER 19

Evals and Observability

Traces, replays, scorecards — the dashboards that keep agents honest

3

metric categories: quality, safety, efficiency

> You cannot govern what you cannot see. Agentic systems that run without comprehensive observability are, from a risk management perspective, essentially black boxes: they produce outputs, they take actions, they consume resources, and the organization has no systematic way to know whether they are doing so correctly, safely, or within the bounds of their governance charter. The observability stack for an agentic program — spanning traces, metrics, replays, and structured evaluations — is not a nice-to-have for mature deployments; it is the prerequisite for responsible deployment at any scale. Without it, the governance charter is a fiction and the policy stack is unenforceable.

Traces: The Atomic Unit of Observability

A **trace** is a structured record of a single agent run from start to finish. It captures every significant event in the run's execution: the initial prompt, each model call and its response, each tool invocation and its result, the intermediate reasoning produced at each planning step, the final output, the total token consumption, the wall-clock latency, and any errors or exceptions that occurred. Traces are the atomic unit of agentic observability because they contain all the information needed to reconstruct what happened during a run and to evaluate whether it went well.

The challenge with traces is their volume and complexity. A production agent handling hundreds of tasks per day generates traces that collectively contain millions of data points — far too much for humans to review manually. The observability infrastructure must therefore provide efficient storage, indexing, and querying of trace data, along with automated analysis that surfaces the traces most likely to be informative: the ones that failed, the ones that were unusually expensive, the ones that triggered escalation, the ones that deviate significantly from the historical baseline.

Langfuse is currently one of the most widely deployed open-source tracing solutions for LLM-based agents. It provides an OpenTelemetry-compatible trace format, a rich web interface for trace exploration, and a scoring system that allows human reviewers to annotate traces with quality judgments that feed back into automated evaluation pipelines. Arize Phoenix offers similar capabilities with a stronger emphasis on statistical analysis of trace populations —

identifying systematic patterns in how the agent's behavior varies across different input types, user groups, or time periods.

Structured Evals: Measuring What Matters

Observability tells you what happened; evaluations tell you whether it was good. A **structured evaluation** is a test of the agent's behavior on a defined set of scenarios, scored against defined criteria. The criteria for agentic evaluations are different from those for conventional model evaluations: in addition to output quality (is the agent's answer correct?), they must include behavioral dimensions that are specific to autonomous systems: tool use appropriateness (did the agent call the right tools in the right order?), instruction-following fidelity (did the agent respect the constraints in its system prompt?), escalation correctness (did the agent escalate when it should have and not escalate when it shouldn't have?), and safety compliance (did the agent avoid prohibited actions even when the task seemed to require them?).

Building effective evaluation suites for agents requires a combination of automated and human evaluation. Automated evaluation using LLM judges — where a second model is asked to score the primary agent's responses on a defined rubric — is scalable and cheap, but it inherits the biases of the judge model and is unreliable for adversarial scenarios. Human evaluation is accurate but expensive and slow. The standard practice is a hybrid: automated evaluation for routine quality metrics, human evaluation for a statistically sampled subset of runs and for all edge cases identified through anomaly detection on the trace data.

Replays and Regression Testing

A **replay** is the re-execution of a historical agent run against a new version of the agent — a new model version, a new system prompt, a new tool configuration — using the same inputs that the historical run received. Replays are the standard mechanism for detecting behavioral regressions before they reach production: if a new model version changes the agent's behavior on a set of historical runs in ways that degrade the evaluation metrics, the regression is detected before the new version is deployed.

Weights & Biases Weave provides a particularly capable replay infrastructure. Its versioning system tracks not just model versions but the full configuration of the agent — system prompt, tool definitions, retrieval settings — and its replay engine re-executes historical runs against any combination of component versions, producing a structured diff of the evaluation metrics between versions. This makes it possible to attribute behavioral changes to specific component updates, which is essential for root-cause analysis when a regression is detected.

The value of replay testing extends beyond regression detection. Organizations that maintain a comprehensive library of historical traces — including traces that document past failures and the correct behavior expected in those situations — have a qualitatively different ability to evaluate new agent versions than those that rely only on synthetic test sets. Historical traces capture the

full diversity of the agent's actual operating environment, including edge cases that would never have been anticipated at test design time.

> *"The organizations that will have the most robust agentic deployments in three years are the ones that are collecting and annotating traces most diligently today. The trace library is the institutional memory of the agent program."*

Scorecards and Dashboards

Individual traces and evaluation runs generate data; scorecards aggregate that data into actionable signals for the governance structure. A well-designed scorecard for an agentic program tracks three categories of metrics. **Quality metrics** — success rate, task completion rate, output quality scores from automated and human evaluations — measure whether the agent is doing its job well. **Safety metrics** — escalation rate, policy violation rate, anomaly detection trigger rate — measure whether the agent is staying within its governance boundaries. **Efficiency metrics** — token consumption per task, latency per task, cost per task — measure whether the agent is operating within the economic parameters of its business case.

The governance charter should specify minimum acceptable thresholds for each category. An agent whose quality metrics fall below threshold should trigger a performance review; an agent whose safety metrics exceed threshold (too many escalations, too many policy violations) should trigger a governance review. The scorecard is the mechanism by which the governance charter's risk appetite is operationalized into a monitoring regime that can run continuously without requiring constant human attention.

> THE EVAL STACK IS NOT OPTIONAL
>
> Observability tooling for agentic systems has matured rapidly: Langfuse, Arize Phoenix, and Weights & Biases Weave are all production-ready, with open-source cores and enterprise licensing tiers. The barrier to building a basic eval stack is no longer tooling availability — it is organizational will. Teams that have not built an eval stack by the time they deploy their first production agent will find themselves making risk management decisions based on intuition rather than evidence, which is not a position that any serious governance charter can endorse.

CHAPTER 20

Guardrails and Policy Engines

From input filters to runtime policy — defense in depth for autonomous systems

4

defensive layers: input, reasoning, output, action

> Governance charters and policy stacks tell the organization what agents are permitted to do; guardrails and policy engines enforce those permissions at runtime. The distinction is not academic. A governance charter that says "this agent shall not send email to external parties without human approval" is worthless unless there is a technical control that actually prevents the agent from doing so — because the agent does not read the charter and the model that powers it has no inherent understanding of organizational policy. Guardrails are the runtime instantiation of governance; without them, governance is aspirational. This chapter maps the full defensive surface from input filters at the conversation boundary to policy engines that enforce constraints mid-workflow.

The Defense-in-Depth Model

Security practitioners have long applied the principle of defense in depth to conventional software systems: no single control is assumed to be perfect, so multiple overlapping controls are layered such that an attacker must defeat several independently to succeed. The same principle applies to agentic guardrails, and it matters more for agents than for conventional systems because the attack surface is wider and more dynamic. An agentic system faces attacks at multiple points: the initial user input may contain a prompt injection; the tool responses retrieved from external services may contain adversarial content; the model's reasoning may be manipulated by carefully crafted context; and the agent's outputs may be intercepted and modified before they reach their destination. A single guardrail that operates at only one of these points provides inadequate coverage.

The layered guardrail model has four principal layers. The **input layer** screens incoming requests before they reach the model, filtering for known attack patterns, PII that should not be included in model prompts, and requests that fall outside the agent's approved use cases. The **reasoning layer** monitors the model's intermediate outputs — its plans, its reasoning traces, its tool call parameters — for indications that it is about to take an action that violates policy. The **output layer** screens the agent's final outputs before they are delivered, checking for policy violations, data leakage, and content that fails defined quality thresholds. The **action layer** enforces policy

at the point of tool invocation — the last line of defense before an action has real-world consequences.

Input Filters and Prompt Injection Defense

Prompt injection is the agentic equivalent of SQL injection: an attacker supplies input that the model interprets as instructions rather than data, causing it to take actions that override its intended behavior. The OWASP Agentic AI Security Initiative has identified prompt injection as one of the top vulnerabilities in agentic systems — and the defense is correspondingly important. Input filters at the conversation boundary should screen for the linguistic patterns associated with known injection attacks: instructions to ignore the system prompt, requests to reveal the system prompt's contents, persona-switching commands, and instructions to perform actions outside the agent's authorized scope.

Input filters are necessary but not sufficient. Indirect prompt injection — where the malicious instruction is not in the user's input but in a document or tool response that the agent retrieves and incorporates into its context — is harder to defend against because it occurs after the input filter has already processed the initial request. Defense against indirect injection requires monitoring at the reasoning layer: the agent's reasoning traces should be analyzed for the pattern of a sudden, unexplained change in goal or behavior that is characteristic of a successful injection attack. This is an active research area with no perfect solution, but several commercial guardrail products have begun offering injection detection at the reasoning layer as a production feature.

Runtime Policy Engines

A **runtime policy engine** is a component that sits between the orchestration framework and the tools, intercepting every tool invocation and evaluating it against a defined policy before allowing it to execute. The policy can be expressed in various forms: a set of allow/deny rules based on the tool's parameters (block any email send to a domain not on the approved list), a statistical anomaly detector (flag any tool call sequence that differs significantly from the historical baseline), or a secondary model call (ask a smaller, cheaper model whether this action is consistent with the agent's approved use case). The runtime policy engine is the action layer of the defense-in-depth model — the control that matters most when everything else has failed.

Policy engine design involves tradeoffs between coverage, latency, and cost. A policy that calls a secondary model for every tool invocation is comprehensive but adds latency and cost to every agent run. A rule-based policy is fast and cheap but brittle — it can only cover scenarios that have been explicitly anticipated. The practical approach for most production deployments is a tiered policy engine: fast, rule-based checks for the most common and most clearly prohibited actions, with secondary model invocation reserved for ambiguous cases that the rules cannot resolve. The threshold for ambiguity should be calibrated to the risk profile of the action: a tool

invocation that could have irreversible consequences should get secondary review even at the cost of additional latency.

> *"A guardrail that adds 200 milliseconds of latency to an agent action that could authorize a $50,000 payment is not a performance problem. It is a bargain."*

Output Screening and Data Leakage Prevention

The output layer is where data leakage prevention controls are most naturally implemented. An agent that has retrieved confidential documents, synthesized their contents, and is about to deliver a summary to a user may be about to deliver information that the user is not authorized to receive — either because the agent made a mistake in its access control logic, or because the user manipulated the agent into retrieving documents that exceeded their clearance level. Output screening checks the agent's response against the classification labels that the data governance framework has assigned to the retrieved content and blocks or redacts the response if it contains material at a classification level above the user's authorization.

Beyond data classification, output screening should also enforce content policies: the agent's responses should not contain hate speech, discriminatory content, or other material that violates the organization's acceptable use policy, even if the model generates it in a context where it might appear to follow naturally from the task. Screening for content policy violations is a well-solved problem for simple question-answering systems, but it is harder for agents because the relevant context is not just the agent's immediate output but the full sequence of actions that led to it — and a content policy violation may be embedded in a tool call parameter rather than in the agent's natural language output.

The Policy-as-Code Paradigm

The most maintainable approach to agentic guardrails treats policy as code: the rules and constraints that govern the agent's behavior are expressed in a machine-readable format, version-controlled alongside the agent's other code, and deployed through the same CI/CD pipeline that deploys changes to the agent's prompts and tools. This policy-as-code paradigm has several advantages. It makes policy changes auditable — every modification is tracked in the version control system with a timestamp, an author, and an explanation. It makes policy testing automated — regression tests can verify that a policy change has the intended effect and no unintended side effects. And it makes policy portable — the same policy definitions can be applied consistently across different deployment environments and different agent implementations.

The practical implication is that policy definitions should be stored in a structured format — YAML, JSON, or a domain-specific language — rather than embedded in natural language in the agent's system prompt. Natural language policies in system prompts are expensive (they

consume context window tokens), brittle (the model may not follow them consistently), and not auditable (there is no version history of when the policy changed and why). Structured, programmatic policy definitions outside the model context are cheaper, more reliable, and more governable.

> THE OWASP AGENTIC TOP 10
>
> The OWASP Agentic AI Security Initiative has published a draft taxonomy of the top security risks in agentic systems. The top three — prompt injection, excessive agency, and insecure tool use — are precisely the threats that the layered guardrail model is designed to address. Teams building guardrail architectures should use this taxonomy as a checklist: if the architecture does not have a specific control for each of the top ten, the gaps should be treated as known risks that require explicit acceptance or mitigation in the governance charter.

CHAPTER 21

The Use Case Portfolio

Pillar III — picking the right work for an agent to do

5
criteria for agent-appropriate use cases

5
clusters in the enterprise use case taxonomy

> The question of which work to give to an agent is, in practice, the most consequential decision in an agentic program — more consequential than the choice of model, more consequential than the choice of orchestration framework, and far more consequential than the choice of vector database. A poorly chosen use case — one where the agent's capabilities don't match the task's requirements, where the risk of autonomous error is too high for the value generated, or where the organization's data and integration readiness is inadequate for what the task demands — will fail visibly and expensively, damaging not just the specific deployment but the organization's appetite for the broader program. A well-chosen portfolio of use cases, by contrast, generates evidence of value, builds organizational capability, and creates the institutional confidence that sustainable agentic programs are built on.

The Portfolio Mindset

Use case selection for agentic AI is a portfolio management problem, not an optimization problem. The goal is not to find the single highest-value use case and execute it perfectly; the goal is to assemble a portfolio of use cases that collectively demonstrates value across multiple dimensions — quick wins that prove the concept, workhorses that generate steady business value, and longer-horizon bets that position the organization for future capability. A portfolio mindset also recognizes that use cases interact: the data prepared for one agent can often power another; the integration work done for one deployment reduces the cost of the next; and the organizational learning generated by one successful agent creates the human capital that enables the next.

The portfolio should be actively managed, not accumulated. A use case that made sense eighteen months ago may no longer make sense today — because the cost structure of agentic AI has changed, because the organization's data readiness has improved and better use cases are now accessible, or because the original deployment has revealed that the task is more complex than it appeared and the agent's performance is not improving toward the required threshold. Regular portfolio reviews — quarterly for a mature program — should assess each use

case against its original business case, flag underperformers for remediation or retirement, and identify new candidates that have become viable as the program's capabilities have expanded.

Taxonomy of Agentic Use Cases

Use cases for enterprise agents cluster into a taxonomy that has emerged from the collective experience of early adopters across industries. **Research and synthesis** tasks — gathering information from multiple sources, summarizing it, and producing a structured output — were among the first enterprise use cases to prove out, because they are relatively forgiving of imperfect accuracy, the output is reviewed by a human before it is acted upon, and the value proposition is clear: the agent can do in minutes what would take a human analyst hours. **Process execution** tasks — following a defined workflow to complete a business process, such as processing an invoice, onboarding a vendor, or generating a compliance report — are higher-stakes but also higher-value, because they displace labor at scale rather than augmenting it.

Monitoring and alerting tasks — continuously watching a data source, identifying events that meet defined criteria, and triggering a response — are a natural fit for agentic AI because the task is repetitive, time-sensitive, and well-defined. **Decision support** tasks — preparing the information and analysis that a human decision-maker needs to make a decision, without making the decision itself — represent the boundary between augmentation and automation: the agent does the legwork, the human retains the judgment. And **interaction management** tasks — handling conversations with customers, employees, or external partners on behalf of the organization — are the most visible and the most sensitive, because errors in these interactions have immediate reputational consequences.

What Makes a Use Case Agent-Appropriate

Not every task benefits from agentic automation. The characteristics that make a task well-suited for an agent can be summarized in five criteria. First, **multi-step structure**: the task requires a sequence of decisions and actions, not a single query-response pair. Second, **tool access requirement**: the task requires accessing external data sources, systems, or services that cannot be incorporated into a single context window. Third, **sufficient error tolerance**: the consequences of an agent error are bounded and recoverable — either because the agent's outputs are reviewed before they are acted upon, or because the actions themselves are reversible. Fourth, **clear success criteria**: it is possible to define and measure what a good outcome looks like, which is necessary for both operational monitoring and governance. Fifth, **data and integration readiness**: the data the agent needs is accessible, clean, and classified appropriately for the agent's permission level.

A task that fails one of these criteria is not necessarily unsuitable for agentic automation — but the failure point should be addressed explicitly before deployment, either by engineering a solution (adding human review to address an error tolerance problem) or by accepting the

residual risk in the governance charter. A task that fails multiple criteria is a strong candidate for deferral until the underlying readiness gaps have been closed.

> *"The question is not whether agents can do the task. With enough prompt engineering, they can do almost anything. The question is whether they should — and whether the organization is ready for them to do it unsupervised at scale."*

Building and Governing the Portfolio

A governed use case portfolio begins with a formal intake process: a lightweight template that captures the task description, the proposed agent architecture, the expected value, the risk assessment, the data and integration readiness status, and the success criteria. The intake process is not a bureaucratic gate — it should take no more than a day to complete and should be designed to help teams think through the questions they need to answer before deployment, not to slow them down. The governance council reviews intake submissions and makes one of three decisions: approve for pilot, defer pending readiness improvement, or decline.

Approved pilots enter a structured governance regime: a named business owner, a technical owner, a data owner, a defined pilot scope and duration, a set of evaluation metrics that will determine whether the pilot advances to production, and a human oversight arrangement that is commensurate with the pilot's risk level. Pilot outcomes feed back into the portfolio management process: pilots that succeed provide evidence for scaling; pilots that fail — and some will — provide the learning that improves the selection and design of subsequent use cases.

> THE PORTFOLIO TRAP: CLUSTERING AT LOW RISK
>
> A failure mode that appears repeatedly in early-stage agentic programs is the tendency to cluster the portfolio around the lowest-risk use cases — the tasks where the agent is least likely to make a costly mistake. This strategy produces a portfolio that demonstrates feasibility but not significance: it shows that agents can do things that are already relatively easy to automate, without demonstrating that they can do the things that would genuinely transform the business. A healthy portfolio should include at least two or three use cases that are genuinely challenging — where the value is large and the readiness work required to unlock it is substantial. These are the use cases that stretch organizational capability and build the long-term institutional advantage that the program is supposed to create.

CHAPTER 22

Value and Feasibility

Scoring use cases on a 2×2 — and the trap of doing the easy ones first

4
value dimensions: efficiency, quality, speed, strategic

3
feasibility assessments: technical, data, integration

> Every use case selection methodology eventually reduces to two questions: how much is it worth, and can we actually do it? The value-feasibility matrix — a 2×2 grid that maps those two dimensions — is the most widely used tool for prioritizing agentic use cases, and it works well for exactly the reason that simple tools usually work well: it forces an explicit conversation about both dimensions rather than allowing the team to optimize for only one. But the 2×2 also contains a trap that swallows a disproportionate number of early agentic programs: the temptation to populate the portfolio entirely with high-feasibility, moderate-value use cases, leaving the high-value, harder-to-execute work for a "later" that rarely arrives on schedule.

Value vs feasibility

A 2×2 every CIO recognizes — and a quadrant most of them get wrong.

FIGURE 22.1VALUE VS FEASIBILITY. LIGHTHOUSES EARN THE NEXT ROUND OF FUNDING; QUICK WINS BUILD LITERACY; STRATEGIC BETS ARE WHY YOU HAVE A COE.

Defining Value for Agentic Work

Value in the context of agentic use cases has multiple dimensions that are not always captured by a single ROI calculation. **Efficiency value** — the reduction in labor cost or time-to-completion

from automating a task — is the most straightforward to quantify and the one that most business cases lead with. **Quality value** — the improvement in the accuracy, consistency, or comprehensiveness of outputs compared to the human baseline — is harder to quantify but often more strategically significant, because it reflects the agent's ability to do things that humans cannot do reliably at scale: checking every document against every policy, monitoring every data feed in real time, maintaining perfect consistency across thousands of parallel decisions. **Speed value** — the reduction in the latency between a trigger event and a completed response — matters enormously in time-sensitive domains like financial trading, cybersecurity incident response, and customer-facing service delivery. And **strategic value** — the extent to which the use case builds organizational capability, generates proprietary data, or creates competitive advantage that compounds over time — is often the least quantified but the most important dimension for long-horizon portfolio planning.

A common mistake in value assessment is to anchor on the cost of the human task being automated. An agent that replaces a task that costs $50,000 per year in labor is creating at most $50,000 per year in value — and usually less, because the human was also doing other things in the time spent on that task. But an agent that enables a task that was previously too expensive to do at all — continuous monitoring across a data corpus that would require fifty analysts to cover, personalized communication at a scale that would require a thousand writers — may create value that is an order of magnitude larger, because it is not replacing existing capacity but creating new capacity.

Measuring Feasibility

Feasibility for an agentic use case is a composite of three assessments: **technical feasibility** (can current agent capabilities reliably complete the task at the required accuracy level?), **data feasibility** (does the organization have the data the agent needs, in the format, quality, and accessibility required?), and **integration feasibility** (can the agent be connected to the systems it needs to interact with, within the security and compliance constraints that apply?). All three dimensions must pass for a use case to be truly feasible, and failure on any one of them typically blocks progress on the others: a technically capable agent that cannot access the data it needs cannot demonstrate its capability; a well-connected agent that generates unreliable outputs cannot build the organizational trust it needs to operate with reduced human oversight.

Technical feasibility is the dimension that is most often overestimated by teams excited about new model capabilities. The right question is not "can the model do this task in a demo?" but "can a production agent do this task reliably enough, across the full distribution of inputs it will encounter, to meet the accuracy threshold required by the business case?" That threshold is almost always substantially higher than what is demonstrated in a prototype, because production inputs are messier, more varied, and more adversarially structured than the carefully curated examples that make demos look impressive.

The Easy-First Trap

The value-feasibility matrix defines four quadrants: high-value/high-feasibility (obvious prioritization), high-value/low-feasibility (worth working to enable), low-value/high-feasibility (quick wins, but not strategically important), and low-value/low-feasibility (avoid). The easy-first trap is the tendency to populate the portfolio with low-value/high-feasibility use cases and call them quick wins, while deferring the high-value/low-feasibility work indefinitely. This produces a program that generates activity without transforming the business.

The trap is reinforced by organizational dynamics that are entirely understandable: quick wins are politically important for sustaining the program's budget and momentum, data readiness and integration work is unglamorous and takes longer than expected, and the teams responsible for high-feasibility deployment often have no direct incentive to solve the readiness problems that would unlock high-value use cases. Breaking out of the trap requires explicit portfolio governance — a rule that at least a defined fraction of the portfolio must be allocated to high-value use cases even when they are currently low-feasibility, with active investment in the readiness improvements needed to raise their feasibility score.

> *"A portfolio of fifty low-risk agent deployments does not produce the same organizational transformation as ten high-value ones. Volume of deployment is not a proxy for strategic impact."*

The Scoring Methodology

Translating the 2×2 into a practical prioritization tool requires a scoring methodology that is simple enough to be applied consistently but rigorous enough to be meaningful. A workable approach scores each dimension on a 1–5 scale, with explicit rubrics for each score level. For value: 1 is a labor cost reduction below $50K annually with no strategic component; 5 is a capability that does not currently exist and creates measurable competitive advantage. For feasibility: 1 means multiple critical gaps in data, integration, or technical capability; 5 means all components are available and the task has been demonstrated to work at the required accuracy in a realistic test environment.

The composite score — value multiplied by feasibility — ranks use cases for near-term prioritization. But the prioritization should not be mechanical: a use case that scores 4×2 (high value, low feasibility) may deserve prioritization over a use case that scores 2×4 (low value, high feasibility) if the organization has a credible plan to close the feasibility gap within a defined timeframe. The scoring methodology creates a common vocabulary for the portfolio discussion; the judgment about which cases to prioritize remains with the governance council.

THE ROI CALCULATION NOBODY DOES

Most business cases for agentic use cases calculate the labor cost of the task being automated and declare that the cost of the agent deployment minus that labor saving is the ROI. This calculation is almost always wrong in two directions simultaneously: it underestimates the value (because it ignores quality, speed, and strategic value) and overestimates the saving (because agent deployment costs — inference, infrastructure, oversight, governance — are higher than most initial estimates assume). A more honest business case models the full cost of ongoing operation, including the cost of the human oversight that the governance charter requires, and values the agent's contribution to quality and speed explicitly rather than treating it as a bonus.

CHAPTER 23

Human in the Loop

Where supervision belongs, where it doesn't, and how to design the handoff

> The phrase "human in the loop" has become a piece of governance vocabulary so widely invoked that it has nearly lost its meaning. Organizations declare that their agentic systems have human oversight; regulators require that high-risk AI systems be subject to human oversight; governance charters specify that certain agent actions require human approval. But "human in the loop" is not a single thing — it is a spectrum of arrangements that range from a human who reads every output before it is used to a human who is nominally available to intervene but in practice never does. The difference between those extremes is the difference between meaningful oversight and a compliance fiction, and the difference matters enormously for both the risk profile of the deployment and the organization's legal standing under instruments like the EU AI Act's Article 14.

The Human Oversight Spectrum

Human involvement in an agentic workflow exists on a spectrum defined by the timing and scope of human intervention. At one end, **human-on-the-loop** supervision: the agent executes autonomously, and a human reviews the outputs after the fact, with the ability to intervene if something is wrong. This arrangement provides genuine accountability but limited risk mitigation: by the time the human reviews, the agent's actions may already have caused harm that cannot be reversed. At the other end, **human-in-the-loop** approval: the agent pauses at defined checkpoints and a human must actively approve before the workflow continues. This arrangement provides strong risk mitigation but adds latency and reduces throughput — it is appropriate for high-stakes, low-volume decisions but unsuitable for high-volume, time-sensitive workflows.

Between these extremes lies a range of intermediate arrangements: conditional approval (the agent proceeds autonomously unless it flags uncertainty or detects a condition that triggers human review), statistical sampling (a random sample of agent outputs is reviewed by humans, with the sample rate calibrated to the risk level), and exception handling (the agent proceeds autonomously unless it encounters an error or an edge case that it has been trained to escalate). Each of these arrangements represents a different tradeoff between oversight effectiveness and operational efficiency, and the right arrangement depends on the specific risk profile of the task, the organization's risk tolerance, and the regulatory requirements that apply.

Designing Effective Checkpoints

A checkpoint is a point in the agent's workflow where execution pauses and a human is asked to review the state and approve (or modify, or abort) the continuation. Checkpoints are only as valuable as their design: a checkpoint that presents the human with an incomprehensible dump of agent state, asks for approval within a ten-second timeout, or is positioned after the agent has already taken the consequential action it was supposed to pause before taking is not a meaningful control. Effective checkpoint design requires getting three things right.

First, the checkpoint must be positioned before the consequential action, not after it. An agent that sends an email and then asks for approval has already done the thing the checkpoint was supposed to prevent. This sounds obvious but is surprisingly often violated in practice, either because the developer did not think carefully about which actions are consequential or because the agent's planning logic re-orders the steps in a way that bypasses the intended checkpoint position. The governance charter's decision authority matrix should specify exactly which action types require a pre-execution checkpoint, and the orchestration framework should enforce those specifications at the workflow level.

Second, the checkpoint must present the human reviewer with the information they need to make a meaningful decision in a reasonable amount of time. A reviewer who is asked to approve an agent action without seeing the reasoning that led to it, the data the agent used, and the expected consequences of approving versus declining is being asked to perform a ceremonial function rather than a genuine oversight function. The checkpoint interface should summarize the agent's current state, its proposed next action, and the key factors that led to it — in plain language, in under a minute of reading time.

Third, the checkpoint must be designed for the expected review volume. A checkpoint that generates ten reviews per day can afford thorough human review of each case. A checkpoint that generates a thousand reviews per day requires a triage system that routes the most important cases to human attention and allows routine cases to proceed with lighter oversight. The triage system itself is a form of automation, which means it must be governed and monitored just like the agent it is supporting.

> *"A checkpoint that no one has time to review carefully is not oversight — it is approval theater. The governance value of a checkpoint is determined not by its existence but by the quality of the reviews it generates."*

The Handoff Protocol

The moment at which an agent hands work to a human — and the reverse, when a human returns work to an agent — is one of the highest-risk points in a hybrid human-agent workflow. At the handoff point, context must be transferred accurately, responsibility must be clearly established, and the receiving party must have the information they need to continue without re-

doing the work that the handing-off party has already completed. Handoffs that fail at any of these three requirements produce gaps in coverage — work that falls between the agent and the human — and conflicts in responsibility where both parties believe the other is handling something that neither is handling.

The handoff protocol should be a formal specification: what information is transferred, in what format, through what channel, with what confirmation that it was received and understood. For agent-to-human handoffs, this typically means a structured summary of the agent's work to date, a clear statement of what action or decision the human is being asked to take, a deadline for the human's response (after which the agent either escalates further or takes a defined default action), and a mechanism for the human to ask clarifying questions and get responses. For human-to-agent handoffs, it means a clear instruction set, a defined scope, and an explicit authorization for the actions the agent is permitted to take.

When to Remove the Human

The long arc of a maturing agentic program is a gradual reduction in the frequency and scope of human-in-the-loop checkpoints as evidence accumulates that the agent can be trusted with more autonomous operation. This reduction should be earned through demonstrated performance — specific evidence that the agent meets defined accuracy thresholds on the classes of action being considered for autonomous operation — not assumed on the basis of general impressions of the agent's capability. The governance charter should specify the evidentiary standard required to reduce oversight for each tier of action: what accuracy threshold must be sustained, over what volume of examples, over what time period, before a checkpoint can be removed or made conditional rather than mandatory.

The removal of a checkpoint is a governance decision, not an operational one. It requires the same three-owner sign-off as the original deployment — business owner, technical owner, data owner — and it should be documented in the charter with a rationale and an effective date. The decision should be revisited when there is evidence that the agent's behavior has changed — a new model version, a significant change in the input distribution, or an anomaly in the observability metrics — because behavior that was trustworthy under the original conditions may not be trustworthy under changed conditions.

> THE ALERT FATIGUE PROBLEM
>
> Human-in-the-loop workflows generate a specific failure mode that security operations teams have long struggled with: alert fatigue. When an agent flags too many actions for human review — because its confidence thresholds are miscalibrated, because its exception-detection logic is too broad, or simply because the volume of work exceeds the review team's capacity — human reviewers begin approving without reading, defeating the purpose of the checkpoint entirely. Avoiding alert fatigue requires tuning the checkpoint triggers as carefully as any other system parameter, with explicit metrics for the review rate, the approval rate, the time-to-review, and the frequency of meaningful human interventions (approvals that modified or rejected the agent's proposed action rather than rubber-stamping it).

CHAPTER 24

From Pilot to Production

The valley of death — and how a use case earns the right to scale

> The valley of death is a well-documented phenomenon in technology adoption: the gap between a promising pilot and a scaled production deployment that destroys more organizational value than it creates. For agentic AI, the valley is notably wide and notably deep. The failure modes that end agentic pilots — data quality problems that were invisible in the curated prototype environment, integration challenges that multiplied as the agent encountered the full complexity of production systems, governance gaps that became visible only when the agent operated at scale, organizational resistance from the teams whose work the agent was supposed to augment — are predictable, and the organizations that cross the valley are those that anticipate them rather than discovering them after the fact.

Why Pilots Succeed and Productions Fail

The most common reason an agentic pilot succeeds is that its conditions are controlled: the data is curated, the inputs are selected, the users are motivated early adopters, and the team is paying close attention and fixing problems quickly. The most common reason the subsequent production deployment fails is that none of those conditions persist. Production data is messier than pilot data by definition — it includes all the edge cases, malformed inputs, and unusual requests that the pilot team carefully excluded. Production users include skeptics, opponents, and people who are actively trying to break the system for reasons ranging from legitimate quality assurance to political resistance. And the team that was fixing problems daily during the pilot is now spread across multiple initiatives, with no time to monitor every production run the way they monitored every pilot run.

The structural response to this pattern is a **production readiness review** that explicitly stress-tests the pilot's design against production conditions before the transition. The review asks: what does the input distribution look like at production scale, and has the agent been evaluated on a sample that represents that distribution? What are the failure modes that have not been encountered in the pilot, and what is the plan when they occur? What does the monitoring and alerting infrastructure look like, and who is responsible for acting on alerts? What is the rollback plan if the production deployment needs to be paused? Organizations that can answer these questions confidently before the transition have materially better production outcomes than those that discover the answers after go-live.

The Integration Cliff

Agentic pilots typically run against a small number of carefully selected integrations — the APIs and data sources that were easiest to connect and most directly relevant to the pilot's task. Production deployment requires connecting to a much larger and more complex set of systems, and the integration work expands non-linearly with the number of systems because integration problems interact: a system that works fine in isolation may behave unexpectedly when it is one of ten systems the agent is calling in parallel, because the systems have dependencies on each other's data, competing rate limits, and inconsistent authentication requirements.

The integration cliff — the point at which the number of required integrations exceeds what the team can build and maintain — is one of the most common causes of production deployment failure. Organizations that approach integration strategically rather than tactically — building and maintaining a documented integration layer with consistent authentication, logging, and error handling rather than building bespoke integrations for each agent deployment — can cross the cliff significantly more easily. The integration spine architecture described in Chapter 25 is the systematic answer to the integration cliff: by standardizing the plumbing, it converts integration from a custom engineering problem to a configuration problem.

Operationalizing Governance at Scale

A governance charter that works for a pilot — where the business owner reads every output and the technical owner reviews the logs daily — typically cannot be scaled to production without significant redesign. The oversight mechanisms that are appropriate for a hundred agent runs per month are not appropriate for ten thousand runs per month, and the human review capacity that is adequate for a pilot is almost never sufficient for a production deployment at scale. Production governance requires three things that pilots typically lack: automated monitoring that can detect problems without requiring human review of every run, tiered oversight that concentrates human attention on the cases where it matters most, and a clear escalation path that is fast enough to be useful when something goes wrong.

The transition from pilot to production is also the moment when the governance charter must be revised from a pilot charter to a production charter. A production charter differs from a pilot charter in three important ways: it names a production operations team in addition to the original three owners; it specifies monitoring thresholds and automated responses rather than just escalation contacts; and it includes a formal capacity planning commitment — the people, the infrastructure, and the budget required to operate and maintain the agent at the projected production scale.

> *"The pilot is a proof of concept. The production deployment is an organizational commitment. The governance gap between those two things is where most agentic programs go to die."*

The Change Management Dimension

Technical production readiness is necessary but not sufficient. Agentic deployments that succeed technically but fail organizationally — because the people whose work the agent is supposed to support do not trust it, have not been trained to use it effectively, or are actively resistant — are common enough to constitute a distinct failure mode. The change management work required to deploy an agent in production is often underestimated because it is less visible than the technical work, but it is typically the more time-consuming and the more consequential.

Effective change management for an agentic deployment requires three investments. First, early and genuine engagement with the people whose work the agent will affect — not to sell them on the deployment, but to understand their concerns, incorporate their expertise into the agent's design, and build the trust that comes from being heard. Second, training that is specific to the agent's actual behavior — not generic AI literacy training but training on what this specific agent does well, what it does poorly, and how to use its outputs effectively. Third, a feedback loop that gives users a structured way to report problems and see those reports acted on — because the users who interact with the agent most are the best source of information about its failure modes, and a deployment that does not capture their feedback is leaving its most valuable quality signal on the floor.

Defining and Measuring Success

An agentic deployment without defined success criteria cannot be declared successful — or failed. Success criteria for a production deployment should be defined before go-live and should include at minimum: a quality threshold (the percentage of agent outputs that meet the accuracy standard required by the use case), a safety threshold (the maximum acceptable rate of policy violations or escalations per unit of work), an efficiency threshold (the maximum acceptable cost per unit of work), and a user satisfaction threshold (the minimum acceptable score on a regular user survey). These criteria become the production charter's performance benchmarks, and the governance council reviews them quarterly to determine whether the deployment continues, scales, or is modified.

> THE THREE CONVERSATIONS YOU MUST HAVE BEFORE GO-LIVE
>
> Before any agentic deployment transitions from pilot to production, three conversations must happen and be documented. First, with legal and compliance: what are the regulatory obligations that attach to this agent in production (as opposed to a pilot, which may have been operating under a research or prototype exemption), and are all of those obligations met? Second, with information security: has the production environment been hardened against the attack vectors that were not present in the pilot, including the full set of integrations and the production user population? Third, with the affected business units: do the people who will be working alongside this agent understand what it does, what it doesn't do, and what they should do when it produces a result they are uncertain about?

CHAPTER 25

The Integration Spine

Pillar IV — APIs, identity, data, and the enterprise systems an agent must touch

3

problems solved by the integration spine: duplication, inconsistency, fragility

> An agent without integrations is a language model with a system prompt — capable of answering questions about its training data but unable to do anything that matters in the real world. The integrations are the agent's limbs: the connections to the systems, services, and data sources that allow it to act. But integrations are also the most common source of production failures in agentic deployments — not because integration is technically difficult, but because the accumulation of bespoke, undocumented, inconsistently secured integration code creates a substrate of fragility and risk that becomes harder to manage as the agent program scales. The integration spine is the architectural antidote: a systematic, standardized, centrally governed layer that treats integration as infrastructure rather than as per-agent customization.

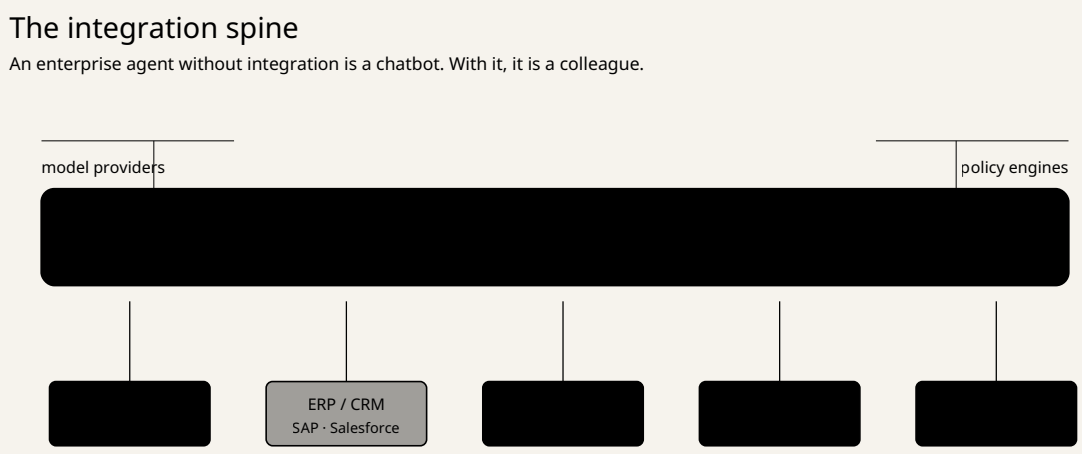

FIGURE 25.1THE INTEGRATION SPINE. FIVE PLUGS AND ONE RUNTIME — EVERYTHING ELSE IS DECORATION.

What the Integration Spine Solves

The integration spine concept addresses a problem that becomes visible at a specific scale threshold: when an organization has more than a handful of deployed agents and the number of integration points has grown beyond what a single team can track and maintain. Below that

threshold, bespoke integration code is a reasonable approach — each agent gets the connections it needs, built by the team that knows the task. Above it, bespoke integration creates three distinct problems. **Duplication**: multiple agents building separate integrations to the same downstream system, each with slightly different authentication logic, error handling, and retry policies. **Inconsistency**: the same system accessed through different integrations with different security configurations, making it impossible to audit the agent program's access profile as a whole. And **fragility**: a change in the downstream system's API breaks multiple integrations simultaneously, with no single owner responsible for fixing them all.

The integration spine solves all three problems by centralizing the integration layer: a single, standardized set of connectors for each downstream system, with consistent authentication, logging, rate limiting, and error handling, maintained by a platform team that is responsible for the entire integration surface. Agent teams consume integrations from the spine rather than building their own, which means they get the security and reliability properties of the spine automatically, and the platform team can make improvements to a single connector that benefit all agents simultaneously.

API Layer and Contract Management

The API layer of the integration spine is the set of interfaces through which agents interact with downstream systems. Each API connection in the spine should be governed by an **API contract**: a formal specification of the API's inputs, outputs, error modes, rate limits, and SLA. The contract is the integration's source of truth — it defines what the agent can expect from the integration and what the integration requires from the agent. When the downstream system's API changes, the contract is updated and the agent teams that depend on it are notified through a formal change management process.

API contract management is particularly important for agentic systems because agents make API calls autonomously, often in sequences and combinations that the integration developer did not anticipate. An agent that encounters an unexpected API error — a rate limit it did not know about, an error response it was not told to handle, a data format that differs from the specification — may fail silently, produce an incorrect output, or enter a retry loop that amplifies the load on the downstream system. A well-managed API contract that accurately documents the system's behavior under all conditions, including edge cases and error states, dramatically reduces the frequency of these failure modes.

Identity and Authentication in the Spine

Authentication is the security-critical dimension of the integration spine. Every agent that calls a downstream system must authenticate to that system with credentials that are appropriately scoped — granting the agent only the permissions it needs for the specific task it is performing, and no more. This is the principle of least privilege applied to integrations, and it requires both an

identity architecture (how agents authenticate to systems) and a credential management infrastructure (how credentials are provisioned, stored, rotated, and revoked).

The identity architecture for agents is discussed in detail in Chapter 26. From the integration spine's perspective, the key requirement is that every API call made through the spine must be associated with a specific agent identity, with a specific permission scope, and with a specific request context that allows the call to be attributed to a particular agent run for audit purposes. This attribution chain — from the API call back to the agent run back to the agent identity back to the human who initiated the task — is the foundation of the forensic trail that governance and compliance require.

> *"The integration spine is the difference between an agent program that scales and one that ossifies. Without it, every new agent adds to a debt load of bespoke integration code that eventually becomes too expensive to maintain and too risky to change."*

Event-Driven Integration and Triggers

The integration spine is not just about agents calling systems; it is also about systems calling agents. An agent that can only be initiated by a human request misses a large class of high-value use cases: automated monitoring, event-driven response, scheduled analysis, and proactive alerting all require the agent to be triggered by system events rather than human instructions. The integration spine should therefore include an event bus — a publish-subscribe infrastructure that allows downstream systems to emit events that trigger agent workflows.

The event bus architecture has important security implications. A system that can trigger an agent workflow by emitting an event is, in effect, an entity with the ability to initiate autonomous action on the organization's behalf. Every event source connected to the event bus must be authenticated and authorized — the event bus should not accept triggering events from unauthenticated sources — and the events themselves should be validated against a schema before they are used to initiate an agent run. An attacker who can inject events into the event bus has effectively bypassed the input filter layer of the guardrail stack, making the event bus one of the highest-priority targets for security hardening in the integration spine.

The Spine as Governance Surface

A well-instrumented integration spine is one of the most valuable governance assets in an agentic program. Because every agent-to-system interaction passes through the spine, the spine's logs constitute a near-complete record of every action every agent has taken in production — which systems were accessed, with what credentials, with what parameters, at what time, as part of what agent run, initiated by what human request. This record is exactly what auditors and incident responders need, and the organizations that have built it at the

infrastructure level — so that it is generated automatically and comprehensively, rather than assembled laboriously after the fact — have a significant advantage over those that have not.

> **PLATFORM TEAM OR AGENT TEAM?**
>
> A common organizational question in agentic programs is whether integration development should be owned by the agent teams (who understand the task requirements) or by a central platform team (who can maintain consistency and security). The answer that experience favors is a hybrid: the platform team owns the integration spine — the core connectors, the authentication infrastructure, the event bus, and the audit logging — while the agent teams own the agent-specific configuration that uses those connectors. This division of responsibility prevents both the fragmentation that results from fully decentralized integration and the bottleneck that results from fully centralized development.

CHAPTER 26

Identity for Agents

Service accounts, scoped tokens, on-behalf-of — auth for non-human actors

3

extensions required for service accounts in agentic systems

> When a human employee logs into a corporate system, their identity is established through a combination of credentials, device trust, and contextual signals — and that identity determines what they are permitted to do. When an agent accesses the same system, none of those signals are available: the agent has no human presence, no device, and no behavioral baseline that a conventional identity system knows how to interpret. Yet the actions the agent takes under an assumed identity can be as consequential as any human action — or more so, given that agents can act at a speed and scale that humans cannot. Identity for agents is not a marginal extension of existing IAM practices; it is a rearchitecting of the identity model for a world in which the actor is autonomous software rather than a human being.

The Problem with Human Credentials

The most common anti-pattern in early agentic deployments is the use of human credentials to authenticate the agent to downstream systems. A developer configures the agent with their own username and password, or their personal API key, and the agent acts with the developer's full permissions in every system it accesses. This is dangerous for several reasons. The credential scope is almost certainly too broad: the developer has permissions that were granted for their human work, not for the specific task the agent is performing. The credential is not rotated on a schedule appropriate for an automated system: a compromised agent with a long-lived human credential gives an attacker persistent, broad access. And the audit trail is contaminated: actions taken by the agent are attributed to the human in the system logs, making it impossible to distinguish the human's actions from the agent's.

The correct approach is to provision a dedicated identity for each agent — not a human identity with reduced permissions, but a **service account** that is designed specifically for autonomous, programmatic use. AWS IAM and equivalent services in Azure and Google Cloud provide the infrastructure for this: roles with explicitly defined permission sets, time-limited credentials that are automatically rotated, and audit logging that attributes every action to the specific role that took it. The service account for an agent should be provisioned with only the permissions

required for the agent's defined scope of action — nothing more — and those permissions should be reviewed and recertified periodically.

Service Accounts and Workload Identity

A service account is a non-human identity — an account in an identity provider that represents a workload rather than a person. Service accounts have existed in enterprise IT for decades, primarily for batch jobs, scheduled tasks, and system-to-system integrations. The adaptation of this model to agentic systems requires three extensions. First, **dynamic scoping**: an agent's permission scope should vary based on the task it is executing, not be fixed at account creation time. An agent processing low-sensitivity documents should have narrower permissions than the same agent processing high-sensitivity documents, even though the underlying identity is the same. Dynamic scoping requires a permission management system that can issue time-limited, task-specific credentials at runtime rather than providing a single set of standing permissions.

Second, **task attribution**: every credential issued to an agent should be associated with a specific task context — the agent run ID, the human who initiated the run, the use case the run belongs to — so that every action taken with the credential can be attributed back through the chain to the human instruction that originated it. This task attribution chain is required by both the EU AI Act's Article 12 logging obligations and the forensic needs of the incident response process. Third, **cross-tenant isolation**: an agent that handles data for multiple customers or organizational units must be prevented from allowing data from one customer to be accessible to another, even within the same agent run. This requires credential isolation between tasks, not just between agents.

OAuth 2.0 and On-Behalf-Of Flows

Many enterprise systems use OAuth 2.0 for authorization, and the agent's interaction with those systems must be mediated through OAuth token flows that express the correct relationship between the agent, the human user, and the resource being accessed. The standard OAuth flows were designed for human users delegating access to applications; agentic systems require a variation that is sometimes called the **on-behalf-of** (OBO) flow: the agent acts on behalf of a named human user, with the human's authorization, but with permissions that are further scoped by the agent's own permission set.

The RFC 8693 token exchange specification provides the technical foundation for OBO flows in agentic contexts: a subject token representing the human user is exchanged for an actor token representing the agent, with the resulting access token scoped to the intersection of the human's permissions and the agent's permitted scope. This token structure preserves the human attribution chain — the access token records both who authorized the action and which agent executed it — while enforcing the principle of least privilege on both dimensions simultaneously. Implementing OBO flows correctly requires cooperation between the identity

provider, the integration spine, and the orchestration framework, which is one reason why identity for agents is best addressed at the infrastructure level rather than in per-agent code.

> *"The identity of an agent is not an afterthought — it is the fulcrum on which the entire access control model balances. Get it wrong, and every other security control in the stack is weakened."*

Credential Lifecycle Management

The credential lifecycle for an agent is different from the credential lifecycle for a human in one critical respect: the agent cannot self-serve. A human whose password is about to expire receives a notification, logs in, and changes it. An agent whose credential is about to expire will either fail silently — producing incorrect outputs because it can no longer access the data it needs — or fail noisily — throwing authentication errors that alert the monitoring system but interrupt the task in progress. Agent credential lifecycle management must therefore be fully automated: credentials should be rotated on a defined schedule, the new credentials should be delivered to the agent through a secrets management system (AWS Secrets Manager, HashiCorp Vault, Azure Key Vault), and the rotation should be transparent to the agent's operation.

Revocation is the other critical lifecycle event. When an agent is decommissioned, its credentials must be revoked immediately — not deprecated, not allowed to expire naturally, but actively revoked so that they cannot be used by an attacker who has retained a copy. The governance charter should specify the maximum time between an agent's decommissioning decision and the revocation of its credentials; for high-privileged agents, that window should be measured in hours, not days.

The Principle of Minimal Identity Surface

The sum of all the identities in an agent program — every service account, every OAuth token, every API key — constitutes the program's identity surface: the attack surface that an adversary can target through credential theft, token injection, or privilege escalation. A program that has minimized its identity surface — through careful permission scoping, short credential lifetimes, and aggressive decommissioning of unused identities — presents a significantly smaller target than one that has accumulated credentials without discipline. The principle of minimal identity surface is the aggregate expression of least privilege: applied not just to individual credentials but to the entire population of credentials that exists in the program at any given time.

THE SERVICE ACCOUNT SPRAWL PROBLEM

Organizations that have deployed agents for more than a year without a systematic identity governance process typically discover, when they conduct a credential audit, that the number of active service accounts significantly exceeds the number of active agent deployments. The surplus accounts are the residue of pilots that ended without proper decommissioning, experiments that were never cleaned up, and integrations that were replaced without revoking the credentials of their predecessors. Each surplus account is a potential foothold for an attacker who obtains its credentials — and the organization typically has no monitoring in place for accounts it has forgotten about. A quarterly credential audit that identifies and revokes all accounts not associated with an active, governed deployment is a minimum hygiene requirement for any serious agentic program.

CHAPTER 27

Data Readiness

Lineage, freshness, classification, and why your data lake is not a vector store

4

dimensions of data readiness: accuracy, freshness, accessibility, classification

> An agent is only as reliable as the data it operates on. This is true of all data-dependent systems, but it is more acutely true of agents because agents act on their data — they do not merely analyze it or report on it. An agent that consults stale inventory data may place an order for parts that are no longer available; an agent that reads a customer record with incorrect contact information will communicate with the wrong person; an agent that retrieves a document without knowing its classification may include restricted content in a response that reaches an unauthorized recipient. Data readiness — the state of data being accurate, current, accessible, and properly classified for the agent's use — is not a precondition for data science work; it is a precondition for autonomous action, and the consequences of inadequate data readiness are correspondingly more serious.

The Four Dimensions of Data Readiness

Data readiness for agentic deployment has four dimensions that must all be addressed. **Accuracy**: the data must correctly represent the entities and states it purports to describe. For agentic use cases, accuracy requirements are often more stringent than for analytical use cases, because an analyst can identify suspicious data and exclude it from their analysis, while an agent acting on that data will not. **Freshness**: the data must be current enough for the decisions the agent is making. An agent performing risk assessments needs data that reflects the current risk state; one performing customer outreach needs data that reflects the customer's current relationship status; one performing compliance monitoring needs data that reflects the current regulatory environment. The freshness requirement varies dramatically by use case and must be assessed explicitly for each deployment.

Accessibility: the data must be reachable by the agent through the integration spine, in a format and at a speed appropriate for the agent's operational requirements. An agent that needs to query a database in real time cannot work with data that is only available in a batch export updated nightly; an agent that needs to retrieve documents from multiple systems cannot work with data that requires manual access requests. Accessibility includes both technical connectivity — the integration exists and works — and latency properties: the data can be retrieved quickly enough for the agent's operational requirements.

Classification: every piece of data the agent can access must carry accurate metadata about its sensitivity level, its permissible uses, and its distribution restrictions. Without this classification metadata, the agent's output-handling logic cannot determine whether a given response is appropriate to deliver to a given recipient, and the data governance framework's controls become unenforceable. Classification is the most frequently underdeveloped dimension of data readiness, because it requires sustained human effort to assign and maintain metadata across large, heterogeneous data estates.

Lineage: Knowing Where Data Came From

Data lineage — the provenance record that tracks where a piece of data came from, how it has been transformed, and what it has been used for — is valuable for conventional analytics but essential for agentic AI. An agent that acts on data cannot simply be held responsible for the quality of its action; the question of responsibility runs upstream to the data itself. If the agent made a wrong decision because it was consulting incorrect data, the question of who is responsible for the data's incorrectness requires a lineage record that can be followed from the agent's decision back to the data source that supplied the incorrect value.

Lineage also matters for regulatory compliance. The EU AI Act's requirements for data governance (Article 10) include obligations to document the data's characteristics, its provenance, and any known data quality issues. Meeting these obligations requires a lineage system that can produce this documentation on demand — not a manual survey that is assembled after the fact, but a continuously maintained record that reflects the actual provenance of the data the agent is using in production. Several data catalog tools — Alation, Collibra, DataHub — now offer lineage tracking that is compatible with agentic data access patterns, though all require integration effort to capture the agent's retrieval activity in the lineage record.

Freshness and the Staleness Risk

Staleness is the data quality failure mode that is most specific to agentic systems, because agents act on data in real time and the consequences of acting on stale data are immediate. An analytical system that uses stale data produces an incorrect analysis that is eventually corrected; an agent that uses stale data takes an incorrect action that may have already caused harm by the time the staleness is detected. The staleness risk varies enormously by data type: financial prices can become stale in milliseconds; customer contact information can become stale in weeks; regulatory guidance can become stale in months. The agent's data architecture must match the data refresh cadence to the staleness tolerance of the use case.

Implementing freshness controls requires instrumenting the data layer: every data retrieval should return not just the data value but the timestamp of its last verified update, and the agent's policy engine should be configured to flag or block actions that depend on data that is older than the staleness threshold for that use case. This sounds straightforward, but many enterprise data

systems do not expose freshness metadata through their APIs, which means the integration work for freshness control often involves modifying or replacing data access patterns that were designed for human analytical use rather than for real-time agentic action.

> *"The data quality that was acceptable for quarterly business reviews is not the data quality required for continuous autonomous action. The upgrade from 'good enough for dashboards' to 'good enough for agents' is frequently the most expensive part of an agentic readiness program."*

RAG and the Retrieval Quality Problem

Retrieval-augmented generation (RAG) — the pattern of augmenting an agent's context with documents retrieved from an external knowledge base — is the standard approach for giving agents access to proprietary organizational knowledge without incorporating that knowledge into model weights. The quality of a RAG system is determined primarily by the quality of its retrieval: a model that would answer correctly given the right context will answer incorrectly if it receives irrelevant or misleading retrieval results. The retrieval quality problem is, in practice, a data readiness problem: it is caused by documents that are outdated, inconsistently structured, poorly indexed, or not classified in a way that makes their content discoverable by semantic search.

Improving retrieval quality for agentic RAG systems requires investment in the **document corpus** — ensuring that documents are current, that outdated versions are deprecated and removed from the retrieval index, and that the indexing pipeline captures the semantic structure of the documents accurately. It also requires investment in the **retrieval evaluation** — measuring retrieval quality with metrics like precision at k and mean reciprocal rank, and using those metrics to tune the retrieval configuration rather than relying on end-to-end agent evaluation as a proxy for retrieval performance.

> DATA READINESS SCORING
>
> A lightweight data readiness scoring framework for agentic use cases rates each of the four dimensions — accuracy, freshness, accessibility, classification — on a 1–3 scale based on the gap between the current state and what the use case requires. A use case that requires all four dimensions at level 3 and currently has them at level 1 is a use case with significant data readiness debt that must be addressed before deployment. Scoring each use case on this framework before it enters the pilot process surfaces the data investment required alongside the technical and integration investment, which is essential for accurate business case modeling and realistic deployment timelines.

CHAPTER 28

MCP, A2A, and the Interop Layer

Model Context Protocol, Agent2Agent, and the protocols that let agents share work

Dec 2025	Jun 2025	6	3
MCP donated to Linux Foundation	A2A donated to Linux Foundation	A2A task lifecycle states	MCP capability types: tools, resources, prompts

> The interoperability layer of agentic AI is in the middle of a rapid, somewhat chaotic standardization process that will determine whether the multi-agent systems being built today are durable or disposable. Two protocols have emerged as the leading candidates for different parts of the interoperability problem. The Model Context Protocol (MCP), originally developed by Anthropic and donated to the Linux Foundation in December 2025, addresses the connection between an agent and the tools and data sources it uses. The Agent2Agent protocol (A2A), originally developed at Google and donated to the Linux Foundation in June 2025, addresses the communication between agents in a multi-agent system. Together they sketch an interoperability stack that has the potential to do for agentic AI what HTTP and TCP/IP did for the web: create a common substrate on which diverse systems can interact without requiring bespoke integration for every pair.

MCP: The Host-Client-Server Model

MCP defines a three-component architecture: a **host** (the application that runs the agent and manages connections), a **client** (a protocol implementation within the host that maintains a connection to a specific server), and a **server** (a lightweight process that exposes tools, resources, and prompts through the MCP interface). The transport between client and server uses JSON-RPC 2.0, a lightweight remote procedure call protocol that is simple enough to implement in any language and already widely supported in developer tooling. An agent that speaks MCP can connect to any MCP server — a file system, a database, a web browser, a code execution environment, a SaaS API — without requiring bespoke integration code for each. The MCP server ecosystem has grown rapidly: as of early 2026, hundreds of MCP servers are publicly available, covering everything from GitHub and Slack to weather APIs and specialized enterprise data systems.

The MCP protocol exposes three types of capabilities: **tools** (functions the agent can call), **resources** (data the agent can read), and **prompts** (reusable prompt templates the agent can invoke). This taxonomy maps cleanly onto the agentic use case taxonomy: tools correspond to the action capabilities an agent needs (send an email, create a record, execute a query);

resources correspond to the data access capabilities it needs (read a document, query a knowledge base, retrieve a configuration); and prompts correspond to the behavioral specifications it needs (follow this procedure, apply this template, respond in this style). The three-category design is not accidental — it reflects a careful analysis of what agents actually need from their environment and creates clean boundaries between capability types that have different security implications.

The security model of MCP has evolved significantly since its initial release. The original version had minimal security specifications; the version donated to the Linux Foundation includes OAuth 2.0 support for authentication and authorization, explicit scope definitions for tool and resource access, and a consent model that requires explicit user authorization for sensitive operations. These security improvements are necessary but not sufficient for enterprise deployment — organizations building production MCP integrations will need to add additional controls at the host layer, particularly around tool call auditing and the enforcement of the decision authority matrix for consequential tool invocations.

A2A: Agent-to-Agent Task Lifecycle

While MCP addresses the agent's connection to its tools and data, A2A addresses the agent's relationship with other agents. In a multi-agent system, agents need to delegate tasks to each other, track the progress of delegated tasks, exchange intermediate results, and handle the completion or failure of a delegated workflow. A2A defines a task lifecycle protocol that standardizes these interactions: a requesting agent sends a task with a structured description and a callback address; the responding agent accepts, processes, and returns results through a defined sequence of status transitions; and the requesting agent tracks the task's progress through a polling or event-streaming interface.

The A2A protocol's task lifecycle has six states: submitted, working, input-required (the receiving agent needs additional information from the requestor), completed, failed, and cancelled. The input-required state is particularly important for enterprise deployments: it models the case where a sub-agent encounters a decision point that requires input from the orchestrating agent or from a human, allowing the task to pause and wait without losing its state. This structured interrupt mechanism is the A2A equivalent of the human-in-the-loop checkpoint that the orchestration stack must support — and its standardization means that multi-agent systems built with A2A get interrupt handling for free, rather than requiring each implementation to build it from scratch.

AGNTCY and the Agent Network Infrastructure

Beyond MCP and A2A, a third initiative is attempting to address the network infrastructure layer that a large-scale, distributed multi-agent ecosystem will require. **AGNTCY** — an open-source initiative led by Cisco with participation from LangChain, CrewAI, and others — is building an "internet of agents": a directory and routing infrastructure that allows agents to discover other

agents, understand their capabilities, and establish secure communication channels with them. Where MCP handles the connection between an agent and its tools and A2A handles the task communication between agents, AGNTCY handles the discovery and routing problem: how does an orchestrating agent find the right sub-agent for a given task among the potentially thousands of agents available in an enterprise's or a network's agent registry?

The AGNTCY project's core components include an **agent registry** (a directory of available agents with structured capability descriptions), an **agent connect** API (a standard interface for initiating A2A connections with registered agents), and an **open agent schema** (a JSON Schema format for describing agent capabilities in a machine-readable way that enables automated agent selection). The combination of AGNTCY's registry, MCP's tool connectivity, and A2A's task lifecycle protocol sketches a complete interoperability stack for multi-agent systems — though as of early 2026, significant gaps remain in the security specifications and the operational tooling for managing large agent networks.

> *"MCP is the USB standard for agent tools. A2A is the TCP/IP for agent communication. AGNTCY is the DNS. Together they are building the plumbing for an internet of agents that nobody has fully imagined yet."*

Comparing MCP and A2A

MCP and A2A are complementary rather than competing protocols, but they do have some functional overlap in the area of agent-to-agent communication that is worth understanding. An MCP server can, in principle, expose another agent as a tool — effectively using MCP to mediate agent-to-agent communication. This is simpler to implement than A2A but loses A2A's structured task lifecycle: MCP tool calls are synchronous request-response interactions, and while the MCP protocol supports streaming, it does not natively model the long-running, interruptible, multi-state task lifecycle that A2A defines. For simple agent-to-agent interactions — an orchestrating agent calling a sub-agent to perform a quick classification or extraction — MCP's tool model may be entirely adequate. For complex, long-running delegated workflows with human-in-the-loop requirements, A2A's task lifecycle model is more appropriate.

The practical implication for enterprise architects is that both protocols are likely to be present in a mature multi-agent deployment, serving different parts of the interoperability problem. MCP provides the rich ecosystem of tool integrations that give agents their capabilities; A2A provides the structured communication protocol that allows those agents to collaborate on complex tasks; and the choice of which to use for any specific agent-to-agent interaction should be made based on the complexity and duration of the interaction rather than as a global architectural decision.

PROTOCOL ADOPTION TIMELINE

MCP was donated to the Linux Foundation in December 2025 and is already widely adopted — major model providers including Anthropic, Google, and OpenAI support it, and the server ecosystem is growing rapidly. A2A was donated in June 2025 and is in active development, with production implementations at several large enterprises. AGNTCY is earlier stage, with its first stable release expected in late 2026. The practical implication for enterprise planning is to build MCP integration now — the ecosystem is mature enough to support production use — and to watch A2A closely for the task delegation and human-in-the-loop capabilities it provides. AGNTCY is worth tracking but not yet ready for production adoption in most enterprise contexts.

PART III

From Assessment to Operating Model

Twelve chapters that turn the model into an operating system: a half-day scorecard, a 30/60/90, a twelve-month roadmap, and what "ready" looks like by 2030.

CHAPTER 29

The Maturity Model

Five levels — Initial, Repeatable, Defined, Managed, Optimized — across all four pillars

value*note*
label

value*note*
label

> Every enterprise that has attempted to introduce a genuinely novel capability — from the first relational database to the first public cloud footprint — has discovered the same uncomfortable truth: the technology arrives years before the organisation does. Agentic AI is no exception. The gap between a working proof-of-concept and a governed, auditable, production-grade agent program is not a gap of model capability; it is a gap of institutional maturity. Naming that gap, measuring it honestly, and charting a path across it is what a maturity model is for.

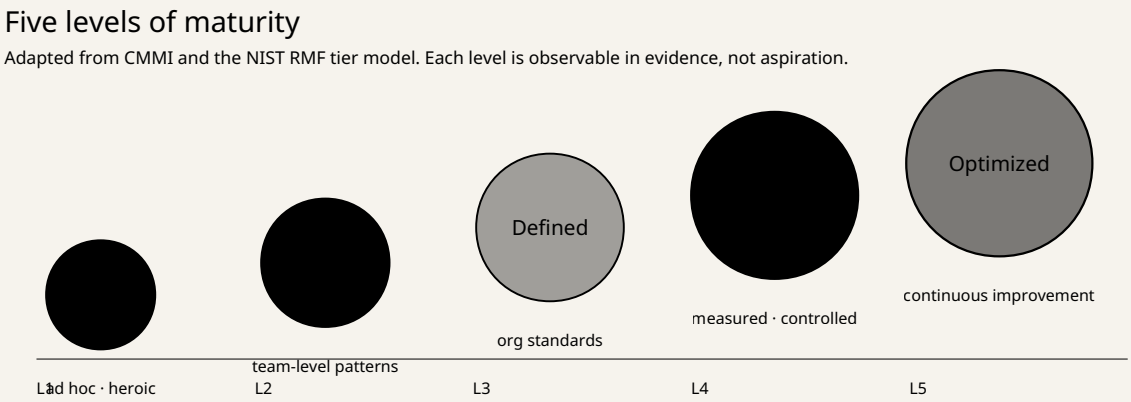

FIGURE 29.1FIVE LEVELS. THE JUMP FROM L2 TO L3 IS WHERE MOST AGENT PROGRAMS STALL — IT REQUIRES STANDARDS, NOT JUST TEAMS.

Why maturity models work — and when they don't

The original insight behind the Capability Maturity Model Integration framework — that processes improve in predictable stages, not in random leaps — has survived three decades of software engineering because it reflects something true about organisations: they do not skip levels. A team that has never written a deployment runbook cannot jump directly to automated rollbacks. A governance committee that has never classified a dataset cannot leap to a real-time

data-lineage graph. Levels exist because the knowledge required for level three is encoded in the failures of level two.

The maturity model presented in this chapter adapts that logic for agentic AI across four dimensions — Governance, Orchestration, Use Cases, and Integration — matching the four pillars introduced in Part I. Each level, from L1 Initial to L5 Optimized, is defined by observable evidence: documents that exist, controls that run, metrics that are tracked. Aspiration does not count. The NIST AI Risk Management Framework takes a similar stance in its tiering model, distinguishing organisations by how systematically they have embedded AI risk practice, not by how sincerely they intend to.

Where maturity models fail is when they become checklists for box-ticking rather than instruments for honest diagnosis. An organisation that reports L4 because it has a policy document, not because it operates the policy, is deceiving its board and imperilling its customers. The antidote is evidence-based scoring: every level claim must be accompanied by the artefact that proves it.

L1 — Initial: the heroic phase

At L1, agentic AI exists because someone cared enough to make it work, not because the organisation made it easy. A team has stood up an agent — perhaps using an API from a frontier model lab and a thin orchestration wrapper — and it is producing results. There is no formal policy governing it. There is no evals harness. The agent's permissions are whatever the developer's credentials allow. If the person who built it leaves, the agent either breaks or becomes ungoverned dark matter in the IT estate.

L1 is not a failure state; it is a beginning. The most important thing an organisation at L1 can do is write down what it has. An inventory of pilots, a rough taxonomy of the risks each presents, and a named owner for each — these are the first deliverables that open the door to L2. Organisations that skip this step discover it later, usually during an incident.

> *"The difference between an experiment and a liability is a changelog and an owner."* — *observation common to every enterprise AI governance review, regardless of industry.*

L2 — Repeatable: patterns emerge at team level

L2 organisations have done one thing L1 organisations have not: they have learned from their first agent and encoded that learning in a repeatable pattern. A team has adopted a shared orchestration framework. A data-handling practice is in place for the agent's memory store. Someone has run a red-team exercise, even informally. The hallmark of L2 is that a second agent can be deployed faster than the first, because the team has retained and shared what the first deployment taught them.

Governance at L2 is still team-level. There may be a policy document, but it was written by the team for the team, and other teams in the organisation do not know it exists. The Gartner Trust, Risk and Security Management framework identifies this as the characteristic failure mode at this level: local competence coexists with enterprise-level blindness. A risk that looks managed from inside the team looks unmanaged from the audit committee's seat.

The transition from L2 to L3 is the hardest in the model. It requires the team to give something up — autonomy, speed, the satisfaction of owning the whole stack — in exchange for an enterprise standard they did not write and may not prefer. This is why most agent programs stall here.

L3 — Defined: enterprise standards arrive

At L3, the organisation has published — and enforces — an enterprise AI policy that applies to all agentic systems, not just the ones built by the team that wrote it. There is a model risk management process adapted for generative and agentic AI. There is an evals standard: minimum test coverage, required red-team scenarios, documented acceptance criteria. There is an identity model for agents: each agent has a machine identity, scoped permissions, and an audit trail tied to a human accountable owner. The ISO/IEC 42001 AI management system standard provides a useful template for what L3 documentation looks like in practice.

L3 organisations typically have a Center of Excellence — even a small one — responsible for maintaining the standard and reviewing new agent deployments before they reach production. The CoE is not a gate designed to slow progress; it is a consistency function designed to prevent twenty teams from making twenty different, incompatible architectural choices that will take three years to unpick.

The measurable outcome of L3 is that an auditor — internal or external — can walk into the organisation, ask for evidence of AI governance, and receive it within a business day. Not a presentation about intentions. Evidence.

L4 — Managed: measurement and control

L4 organisations do not just have standards; they track compliance with those standards in real time. An evals dashboard shows pass rates across production agents. A model risk register is live, with open items assigned to owners and tracked to closure. The organisation can answer, at any moment, how many agents are running, what permissions each holds, what their last eval score was, and who is accountable for each. This is not bureaucracy; it is the minimum information a board needs to discharge its fiduciary duty under frameworks like the EU AI Act.

At L4, the organisation has also developed feedback loops between production performance and the policy stack. When an agent behaves unexpectedly in production, the incident feeds back into the red-team library, the evals harness, and the policy. The organisation gets smarter from failures rather than merely recovering from them.

L5 — Optimized: continuous improvement as culture

L5 is not a destination; it is a disposition. Organisations at this level treat the maturity model itself as a managed asset: they run annual self-assessments, benchmark against peers, and revise their standards when the technology or the regulatory landscape shifts. Their evals library is continuously expanded. Their identity and access model evolves with new agent architectures. Their procurement playbook is updated every time a new class of vendor risk emerges.

Critically, L5 organisations export knowledge. They contribute to industry working groups, publish incident post-mortems (anonymised where necessary), and help raise the floor for the sector. This is not altruism; it is risk management. A sector in which the weakest players are at L1 creates systemic risk that eventually lands on the strongest players too.

> **Diagnostic question for every level:** if your most knowledgeable AI engineer left tomorrow, would your agent program stay at the same maturity level or drop by one? If the answer is "drop by one," you are operating a level below where you believe you are.

Reading the model across all four pillars

The full model produces a 4×5 matrix — four pillars, five levels each. Most organisations are not at the same level across all four. A financial services firm may be at L4 on Governance (long accustomed to model risk management) and L1 on Orchestration (agents are new, frameworks are immature). A SaaS company may be at L3 on Orchestration and L1 on Governance (engineers lead, lawyers follow). The profile is as diagnostic as any single level score.

The next chapter introduces the Scorecard Method: a structured half-day exercise that produces this profile from evidence, not self-report. The chapter after that explains how to read the resulting radar and decide where to invest first. Together, these three chapters form the diagnostic engine at the heart of Part III.

CHAPTER 30

The Scorecard Method

How to assess your organization in a half-day, with evidence

> Assessment frameworks fail not because they ask the wrong questions but because they take too long to answer. A three-month readiness review arrives three months too late; the pilots have already launched, the architecture is already set, and the governance gaps are already baked in. The Scorecard Method is a deliberate counter-proposal: a structured half-day exercise that produces a scored, evidence-backed four-pillar profile usable by an executive on the day it is run.

The half-day structure

The exercise runs in four two-hour blocks, one per pillar, with a facilitator and three to five subject-matter experts per block. The facilitator brings a question set — twenty to twenty-five scored items per pillar, calibrated to the five maturity levels — and the discipline to accept only one form of response: evidence. Opinions are noted but not scored. What counts is the document, the dashboard, the ticket, the log.

Block one covers Governance: does an AI policy exist and apply to agents? Is there a model risk owner? Has any agent been through a red-team exercise? What is the incident response process for an agent failure? Block two covers Orchestration: which framework is in use? Is there an evals harness? How are agent identities issued and rotated? Is there an observability layer that logs every tool call? Block three covers Use Cases: how many agents are in production? How were they selected? Is there a value-feasibility scoring process? What is the kill criterion for a failing agent? Block four covers Integration: is there an API catalogue? How are secrets managed for agent credentials? Is there a data-classification layer that governs what the agent can read?

Each block ends with a consensus score — one of five levels — for that pillar, agreed by the participants and signed by the most senior person in the room. The facilitator synthesises the four scores into a radar profile at the end of the half-day.

Evidence standards

The most important discipline in the Scorecard Method is the distinction between evidence and assertion. An assertion is a claim that something exists or is done. Evidence is the thing itself, or a reliable proxy. The NIST AI Risk Management Framework distinguishes between "govern," "map," "measure," and "manage" — not as aspirational verbs but as observable states. The Scorecard adopts the same logic: for each question, the facilitator must be able to specify what evidence would justify a "yes."

A useful heuristic: if an auditor arrived tomorrow with a subpoena, would the evidence hold up? A policy document that was last reviewed in 2022 and has never been tested in an incident does not justify an L3 Governance score. A policy document that was triggered by a real event, produced a documented response, and was updated afterward does.

> *"The question is not whether you have a policy. The question is whether the policy has ever been tested." — paraphrased from a Forrester AI governance workshop, 2024.*

Scoring calibration

Inter-rater reliability is the Scorecard's most common failure mode. Two facilitators, running the same organisation independently, should produce scores within one level of each other. In practice, facilitators who are not calibrated can diverge by two levels on Governance alone, because "policy exists" means different things to different people.

The solution is an anchor library: for each maturity level on each pillar, a set of exemplar descriptions drawn from real organisations (anonymised), so that each level is associated with a concrete picture rather than an abstract definition. The anchor library is not included in this volume, but the interactive scorecard at the companion site provides worked examples for each question.

A second calibration mechanism is the mandatory disagreement round. Before the block score is finalised, the facilitator must surface any participant who would score differently and record their reasoning. This prevents false consensus and often surfaces the most important information of the session — the gap between what the organisation believes it does and what it actually does.

What comes out

The output of a Scorecard session is four artefacts. First, the scored radar profile: a four-pillar chart, with each pillar at one of five levels, representing the organisation's current state. Second, the evidence log: a record of which evidence was presented for each question, and what gaps were identified. Third, the priority list: the two or three questions where the gap between current level and the next level is smallest and the cost of closing it is lowest — the quick wins. Fourth, the risk register delta: any item that, in the facilitator's judgment, represents an acute risk regardless of maturity level — a running agent with no owner, a production deployment with no kill switch, a data access scope that would alarm a regulator.

MIT CISR research on digital transformation readiness consistently finds that the organisations that improve fastest are not those with the highest initial scores but those with the most honest initial assessments. The Scorecard is an instrument for honesty, not for comfort.

> **Common shortcut — and why it fails:** some organisations attempt to run the Scorecard as a survey, asking thirty people to rate their pillar online and averaging the scores. The result is a political document, not an assessment. The half-day format works because it forces disagreement into the open, where it can be resolved with evidence.

Repeating the Scorecard

The Scorecard is most valuable as a repeated instrument, not a one-time event. Run at baseline, then at six months and twelve months, it produces a trajectory. A program that improves by one level on two pillars in twelve months is making real progress. A program that stays flat on Governance for two years, despite investment in Orchestration and Use Cases, is accumulating risk that will eventually concentrate into an incident.

The twelve-month roadmap in Chapter 33 is structured around Scorecard checkpoints. The operating model decision in Chapter 34 is informed by which pillar the Scorecard identifies as the limiting constraint. The Scorecard is not a standalone exercise; it is the diagnostic instrument that calibrates every subsequent decision in this part of the book.

★ INTERACTIVE SCORECARD

Score Your Organization

Score yourself across four pillars · five levels · 40 questions

> The interactive scorecard lives on the web — radar chart, auto-generated 30/60/90 roadmap, and a shareable URL hash. Visit the chapter online at **chapters/scorecard.html**. The forty questions, four per pillar level, are listed in chapter 30 ("The Scorecard Method"). Run the assessment in a half-day with the four pillar leads in the room; export the snapshot; revisit on a quarterly cadence.

CHAPTER 31

Reading the Radar

What a four-pillar profile tells you about where to invest first

value*note*

label

> A four-pillar profile is a picture, not a prescription. The picture shows you where you are; reading it well tells you where to go next. Most organisations make the mistake of investing in their strongest pillar — the one where the team is confident, the roadmap is clear, and the wins are visible — when the right move is almost always to invest in the weakest. The radar tells you which dent is load-bearing.

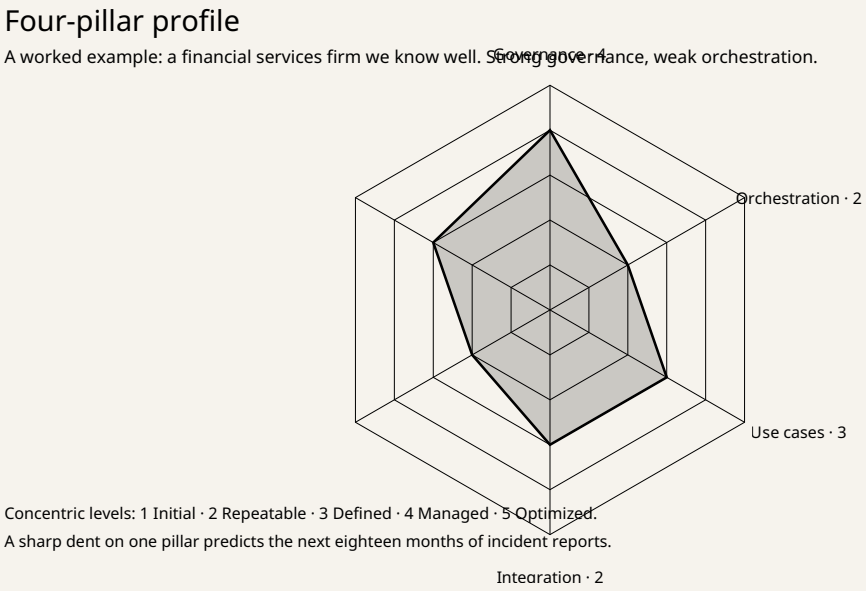

FIGURE 31.1A FOUR-PILLAR PROFILE, PLOTTED ON A RADAR. SHARP DENTS PREDICT WHERE THE NEXT YEAR'S INCIDENTS WILL COME FROM.

Reading the shape

A balanced radar — all four pillars at roughly the same level — is rare, and not necessarily the goal. What matters is the relationship between the pillars. Certain imbalances are benign; others are dangerous. A firm that scores high on Governance and low on Use Cases has a policy apparatus in search of something to govern — a manageable problem. A firm that scores high on Orchestration and low on Governance has built a machine that can act faster than it can be stopped — a dangerous one.

The radar in Figure 31.1 shows a representative profile from a financial services firm: strong Governance (L4), moderate Use Cases (L3), weak Orchestration (L2), weak Integration (L2). The interpretation is immediate: this organisation's agents are well-governed in principle but under-instrumented in practice. The policy says agents must log every tool call; the orchestration layer does not actually do so. The policy says agent identities must be scoped; the integration layer issues credentials with excessive scope because the identity model for agents has not been built. The risk is not that the governance is weak — it is that the governance is strong on paper and weak in execution.

Pillar interaction effects

The four pillars interact. A Use Case portfolio that is rated at L4 — well-scored, well-governed, with clear kill criteria — is only as safe as the Orchestration layer running those use cases. If the Orchestration pillar is at L2, the kill criteria exist in a document but not in a circuit breaker. Similarly, an Integration pillar at L3 — APIs catalogued, secrets managed, data classification in place — becomes a liability if Governance is at L1, because the agent can reach data that no policy has decided it should be allowed to reach.

The OWASP Agentic AI Security Initiative documents this interaction effect in its threat model: the highest-risk scenarios are not those where one pillar is weak but those where two adjacent pillars are mismatched. A strong Use Case selection process (L3) running on a weak Orchestration layer (L1) is the setup for a prompt-injection incident. A strong Integration spine (L4) with weak Governance (L1) is the setup for a data-exfiltration incident. The radar reveals these mismatches at a glance.

McKinsey's State of AI research consistently finds that enterprises with the highest returns from AI investment are those that develop governance and technical capabilities in parallel, rather than sequentially. The radar is the instrument that keeps those two tracks synchronised.

The six archetypal profiles

Across dozens of enterprise assessments, six recurring profile shapes emerge. The Compliance-first profile (high Governance, low everything else) is common in regulated industries: the legal team arrived before the engineers. The Engineer-first profile (high Orchestration, low Governance) is common in technology companies: the engineers arrived before the legal team. The Pilot-heavy profile (high Use Cases, low Orchestration and Integration) is common in consulting-influenced organisations: the strategy was clear, the execution was not. The Data-rich profile (high Integration, low Use Cases) is common in data-mature organisations that have yet to identify what the agent should actually do with all the data it can reach. The Balanced-low profile (all pillars at L1–L2) is the most common of all: the organisation has started but has not yet committed. The Balanced-high profile (all pillars at L3–L4) is the rarest and the most valuable.

> **The profile that looks best but isn't:** a Governance-only high score is the most dangerous false positive in the model. It gives boards comfort that nothing catastrophic can happen while the actual controls are absent. If you are scoring L4 on Governance and L1 on Orchestration, your governance is decoration.

Investment sequencing from the radar

The practical output of a radar reading is an investment sequence: where to spend the next six months, and in what order. The sequencing heuristic is straightforward: close the most dangerous gap first, then the gap that blocks the most value. The most dangerous gap is defined by the interaction-effect analysis above — the two-pillar mismatch that creates acute risk. The most value-blocking gap is the pillar whose low score is holding down the ceiling for every other pillar.

In the financial services example from Figure 31.1, the most dangerous gap is Governance/Orchestration mismatch — paper policies with no runtime enforcement. The first investment should be in an orchestration layer that enforces the existing policy automatically: structured logging, circuit breakers, scoped identities. This does not require new policy work; it requires implementing the policy already on paper. Once the mismatch is closed, the integration investment follows: agent credentials scoped to the data classification already defined by Governance.

The radar reading also reveals what not to invest in. An organisation at L2 across all four pillars that is tempted to acquire an enterprise AI platform at L4 sophistication will find it unusable: the organisation lacks the governance, the evals culture, and the integration plumbing to operate the platform. Tools are only as good as the maturity level of the organisation operating them.

The radar as communication tool

Beyond its diagnostic function, the radar is the most effective communication tool available for briefing a board or executive committee on the state of an agent program. Four pillars, five levels, one picture — it takes thirty seconds to read and generates the right questions. Where are we weakest? What would it take to close the gap? What is the risk of not closing it?

Chapter 33 structures the twelve-month roadmap explicitly around radar checkpoints: a baseline at Q0, a progress read at Q2, and an expected target state at Q4. The roadmap is not a project plan in isolation; it is a commitment to move a specific set of pillars by a specific number of levels in twelve months, with the evidence to prove it.

CHAPTER 32

The 30/60/90

What an enterprise can credibly accomplish in the first ninety days

value*note*
label

value*note*
label

> The ninety-day plan is one of the most abused instruments in management. Executives produce them for onboarding, for strategy reviews, for board decks — and then file them. What makes a ninety-day plan for an agentic AI program different is that it is not a communication document; it is a commitment device. It names specific deliverables, assigns them to named owners, and creates accountability to a board or steering committee that can verify completion. Done right, ninety days is enough time to establish the institutional foundations that will determine whether the next three years of the program succeed or fail.

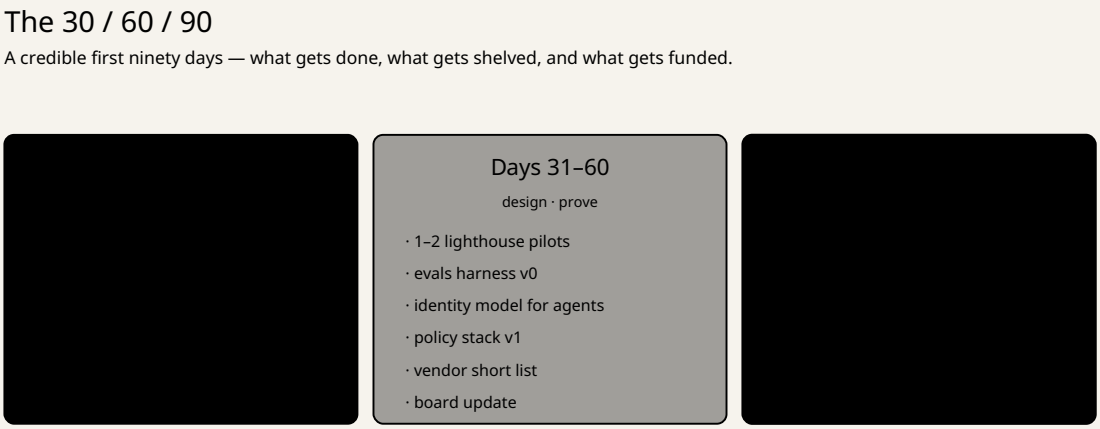

FIGURE 32.1NINETY DAYS. BY DAY 90 YOU HAVE EITHER EARNED THE RIGHT TO SCALE OR LEARNED WHY YOU SHOULDN'T.

Days 1–30: Orient, inventory, charter

The first thirty days are not for building. They are for learning what already exists and establishing the authority to govern it. Three deliverables are non-negotiable. First, the **programme charter**: a two-page document, signed by the most senior executive sponsor available, that names the governance owner, defines the scope of the programme (which agent deployments are in scope, which business units, which risk categories), and commits the

organisation to the Scorecard baseline and twelve-month roadmap. A charter without a named governance owner is an aspiration, not a commitment.

Second, the **NIST RMF gap assessment**: a structured review of the organisation's current AI risk management practice against the four functions of the NIST AI Risk Management Framework — Govern, Map, Measure, Manage. The gap assessment does not need to be exhaustive; it needs to be honest. Which of the four functions does the organisation currently perform, at what tier, and what is missing? This document becomes the governance backlog that the programme will work down over the following sixty days.

Third, the **governance owner named and briefed**: a specific individual, with the authority and the budget to make decisions, who accepts accountability for the programme's governance outcomes. Without this, every subsequent decision becomes a negotiation between teams that each have partial authority. The governance owner is not the most senior AI person in the organisation; they are the person most capable of saying no to a deployment that fails the risk bar, and being heard when they do.

> *"The first thirty days should produce three sheets of paper and one conversation — the conversation where someone accepts accountability." — adapted from MIT CISR digital transformation research.*

Days 31–60: Baseline, scorecard, use-case scoping

With a charter and a governance owner in place, the programme can now make its first honest assessment. The **pillar-by-pillar baseline** is produced using the Scorecard Method from Chapter 30: a half-day facilitated session per pillar, producing a scored radar and an evidence log. This baseline is the contract against which progress will be measured. Every organisation discovers something uncomfortable in this phase; the value is in the discovery, not in the score.

The baseline feeds directly into the **first three use cases scoped**. Scoping here means not selecting which three agents to build but deciding which three agent opportunities the organisation is prepared to govern, instrument, and evaluate. Each use case must have: a named business owner, a stated success metric, a defined eval set (even a small one), an identified data source with its classification, and a kill criterion — the specific condition under which the pilot will be terminated. Use cases without kill criteria are not pilots; they are commitments.

The third deliverable of this phase is the **scorecard run** as a shared document, distributed to the steering committee and reviewed in a sixty-day check-in. The check-in is not a progress presentation; it is a gap-closure negotiation. For each pillar where the score is below the target, the governance owner must present the specific action and the owner responsible for closing it by day ninety.

Days 61–90: First pilot, evals dashboard, roadmap signed

The final thirty days are about demonstrating that the institutional foundations built in days one through sixty are real enough to support a production agent. The **first guarded pilot in production** does not need to be impressive; it needs to be governed. An agent that processes a narrow, well-defined task with full audit logging, a documented eval set with a passing score, a scoped machine identity, and a named human accountable for every action the agent takes is worth more than ten pilots that are impressive and ungoverned. The first production pilot is the proof of concept for the governance model, not just for the agent.

The **evals dashboard** is the instrumentation that makes the pilot's governance visible. At minimum: a pass rate for the eval set, a latency distribution, a log of every tool call, and a flag for any action that exceeded the agent's defined permission scope. The dashboard need not be sophisticated — a spreadsheet updated daily is sufficient for a guarded pilot — but it must exist and must be reviewed by the governance owner at least once a week.

The **twelve-month roadmap**, introduced in Chapter 33, must be signed off by the steering committee before day ninety closes. Signed off means more than approved: it means budgeted, staffed at least at the CoE seed level, and tied to a specific set of pillar-level targets on the Scorecard that the programme commits to reach by each quarter. A roadmap that is reviewed but not funded is a wish list.

> **The most common day-ninety failure:** a programme that has completed all three phases but not produced a single line of evidence for the governance owner. The governance owner has been named but has not reviewed anything. The charter has been signed but has not been tested. The evals dashboard exists but has not been actioned. Ninety days of paper is not ninety days of maturity.

What not to do in ninety days

The list of things that a ninety-day plan should not include is as important as what it should. Do not select an enterprise platform before the Scorecard baseline is complete: the platform choice should follow the maturity gap, not precede it. Do not attempt to deploy more than one guarded pilot in the first ninety days: the governance model is being proven for the first time, and a second pilot running in parallel doubles the complexity before the model is validated. Do not commission an external consultant to write the governance policy: policy that is written externally and owned internally by nobody is the single most reliable predictor of L1 Governance scores eighteen months later.

Ninety days is also not enough time to train a workforce, stand up a full CoE, complete an ISO 42001 readiness review, or select and onboard an orchestration vendor at enterprise scale. These are Q2 and Q3 activities, and attempting them in Q1 is how programmes run out of momentum before they have a win to show for it. The discipline of the ninety-day plan is knowing what to defer.

CHAPTER 33

The Twelve-Month Roadmap

From scattered pilots to a governed agent platform — by quarter

> A roadmap is not a plan; a plan describes activities. A roadmap describes the state the organisation will be in at each milestone — what it will be capable of that it cannot do today, what risks will be managed that are currently open, what evidence will exist that does not exist now. The twelve-month roadmap for an agentic AI programme is, in this sense, a maturity commitment: a promise to the board, by quarter, of where each of the four pillars will land on the Scorecard. The pilot programme that cannot make this commitment credibly is not a programme; it is an experiment.

Q1: Charter, scorecard, three pilots picked, NIST RMF baseline

Q1 is the ninety-day plan from Chapter 32, executed well. By the end of the first quarter, the programme has a signed charter, a governance owner, a Scorecard baseline with a four-pillar radar, and three use cases that have passed the scoping filter — named owner, success metric, eval set, data classification, kill criterion. The NIST AI RMF gap assessment has been completed and the resulting action list is in the governance owner's backlog, prioritised by risk.

The NIST RMF baseline is particularly important for Q1 because it sets the regulatory posture before any production deployment. Organisations operating under the EU AI Act need to know, by Q1, whether any of their planned agent deployments fall under the high-risk categories defined in Annex III. Discovering this in Q3, after a deployment is live, is significantly more expensive than discovering it in Q1, when the design can still be changed.

> *"Q1 is for learning. Q2 is for proving. Q3 is for scaling. Q4 is for owning." — an internal framing used by the AI governance team at a large European financial institution, paraphrased.*

Q2: Pilot 1 to production, orchestration choice, evals platform

Q2 is the first real test of the programme's governance model. Pilot 1 reaches production — not a demo environment, not a limited beta, but a live system handling real work with real consequences. It is accompanied by guardrails: a circuit breaker that stops the agent if it exceeds its permission scope, a structured eval set with a minimum pass rate that must be maintained for the pilot to stay live, and a runbook for the incident response procedure if the agent behaves unexpectedly.

The **orchestration choice** should be made in Q2, not Q1. The Q1 pilots were small enough to run on whatever framework the engineering team preferred; by Q2, the programme needs a single orchestration standard that all future agents will use. This decision — LangGraph, AutoGen, a proprietary platform, or a bespoke internal framework — is one of the most consequential technical decisions in the programme, and it should be made with the benefit of one quarter of operational experience, not in the abstract. Key criteria: does the framework produce structured audit logs? Does it support human-in-the-loop interruption? Does it integrate with the organisation's existing identity and secrets management infrastructure?

The **evals platform** is the instrumentation that will govern every future agent. At minimum: a test runner that executes the eval set against the production agent on each deployment, a dashboard that makes pass rates visible to the governance owner, and a gating mechanism that blocks deployment if the pass rate drops below the agreed threshold. The evals platform is not an optional quality-assurance measure; it is the operational foundation of the governance model.

Q3: ISO 42001 readiness, Pilots 2–3, agent identity model

Q3 is the scaling phase. With one production agent running and a governance model proven, the programme can now expand to the second and third pilots while simultaneously investing in the institutional infrastructure that will make scaling sustainable. The two activities are not independent: each new pilot exercises the governance model in a new domain, producing evidence for the ISO/IEC 42001 readiness review that should be in progress or complete by the end of the quarter.

ISO 42001 is the natural target standard for an enterprise that wants a third-party–validated AI management system. The readiness review — conducted against the standard's requirements for AI policy, risk assessment, performance evaluation, and continual improvement — will identify the remaining gaps between the programme's current state and a certifiable AI management system. For most organisations that have followed the Q1 and Q2 roadmap faithfully, the gaps at this point are administrative rather than structural: documentation of decisions already made, formalisation of practices already running.

The **agent identity model** is the Q3 infrastructure investment that most programmes underestimate. Every production agent needs a machine identity: a credential that is tied to the agent's specific role, scoped to the data and tools the agent is authorised to use, rotated on a schedule, and auditable. Without a formal identity model, agents accumulate permissions informally — developers use personal credentials, or a single service account is shared across multiple agents — and the resulting access graph becomes impossible to audit. Building the identity model in Q3, before the agent estate grows large, costs a fraction of what it costs to retrofit it in Q4 or year two.

> **The Q3 trap:** the programme is running, the first pilot is producing visible results, and the temptation is to launch everything at once. Three more pilots, a new orchestration vendor, an ISO audit, and a change management programme — all in the same quarter. This is how programmes lose the governance discipline they spent Q1 and Q2 building. Pick two. Do them well.

Q4: Operating model decision, procurement playbook, first annual review

By Q4, the programme has enough operational experience to make the operating model decision that will govern how the organisation scales agent deployments beyond the initial programme. The three models — Center of Excellence, hub-and-spoke, and federated — are described in detail in Chapter 34. The Q4 decision is not which model to adopt eventually; it is which model the organisation will commit to for the next two years, with the governance infrastructure to support it.

The **procurement playbook** codifies everything the programme has learned about buying agent-adjacent technology: model provider contracts, orchestration platforms, evals vendors, security tooling. It includes the contract clauses the programme has learned it needs — data residency, incident notification windows, model version pinning, exit rights — and the vendor risk assessment process for agent-specific supply chain risk. Chapter 36 covers this in depth; the Q4 roadmap milestone is to have the playbook written and reviewed by legal before the programme's next major procurement decision.

The **first annual review** is the most important governance event of the year. It is not a retrospective on activities; it is a re-run of the Scorecard, producing a new four-pillar radar that can be compared directly to the Q1 baseline. The comparison is the evidence the board needs to assess the programme's progress: which pillars moved, by how much, and what evidence supports the scores. Any pillar that has not moved since Q1 is a question the board should ask, and the governance owner must have an answer.

CHAPTER 34

The Operating Model

Center of excellence, hub-and-spoke, federated — pick one and live with it

> The question of how to organise an agentic AI programme is not, at its root, a technology question. It is an authority question: who decides what an agent is allowed to do, and who is accountable when it does something it should not? The operating model is the answer. Three models have emerged in enterprise practice — the Center of Excellence, hub-and-spoke, and federated — each embodying a different answer to the authority question, and each making a different set of trade-offs between speed, safety, and scale.

The Center of Excellence

In a Center of Excellence model, a central team owns the agent platform, the policy stack, the evals framework, and the vendor relationships. Business units bring use cases to the CoE; the CoE assesses, builds, deploys, and operates them. The authority structure is clear: the CoE says yes or no, and its no is final. This model produces the most consistent governance outcomes and the fastest accumulation of institutional knowledge. It also produces the most friction, the longest queues, and the most frustrated business units.

The CoE model is appropriate in three situations: the organisation is early in its maturity journey (L1–L2 overall); the risk profile is high (financial services, healthcare, critical infrastructure); or the organisation has a history of decentralised technology decisions producing expensive inconsistencies. The CoE is the right answer when the cost of a governance failure outweighs the cost of a slower deployment cycle.

MIT CISR research on enterprise technology governance consistently finds that organisations with high levels of IT centralisation make more consistent risk decisions and worse innovation decisions. The CoE for agentic AI inherits this trade-off. Its job is not to maximise agent deployments; it is to maximise the quality of agent deployments — a subtly different brief.

Hub-and-spoke

The hub-and-spoke model distributes execution while centralising standards. The hub — typically the CoE or a platform engineering team — owns the architecture, the policy templates, the identity infrastructure, and the evals platform. The spokes — business unit AI teams, often called guilds or chapters — own the use-case selection, the domain data, and the day-to-day operation of agents within their business unit. The hub sets the floor; the spokes build above it.

This is the model most enterprises at L3–L4 maturity gravitate toward, because it balances the competing pressures well. Business units move at the speed their domain demands. The hub maintains the institutional knowledge and prevents the fragmentation that kills federated models. The authority is shared: the spoke cannot override the hub's security controls, but the hub cannot block a deployment that meets the standard. Gartner's TRiSM framework recommends a hub-and-spoke structure as the target state for enterprise AI governance at scale.

> *"The hub does not tell the spokes what to build. The hub tells the spokes what they must never skip — and then gets out of the way." — observed at multiple enterprise AI governance workshops, 2024–2025.*

The federated model

In a federated model, business units own their agent programmes end-to-end: policy, architecture, evals, vendor selection, and operation. A corporate AI policy exists, but it is implemented differently in each business unit, and there is no central authority with the power to halt a deployment. The federated model moves fast and breaks things — not metaphorically, but literally, because the absence of a shared standard means each business unit learns the same lessons independently and pays for the same mistakes independently.

Federated is the model that emerges when no one makes the hub-and-spoke decision: it is the default, not a choice. Most organisations that describe themselves as federated are organisations that did not have a CoE early enough. The federated model can work at high maturity — if every business unit is genuinely at L4 or L5 across all four pillars — but reaching that state through federation takes significantly longer than reaching it through a hub-and-spoke with a strong hub.

> **The model you're already in:** most organisations reading this chapter are already in a de facto federated model. Three business units have agents running. Two of them used different frameworks. None of them has an evals harness. The question is not which model to choose from scratch but which model to migrate toward — and migration to hub-and-spoke is significantly easier than migration to CoE.

Staffing the hub

A hub-and-spoke model is only as good as its hub. A hub of two people managing an estate of thirty agents across eight business units is not a hub; it is a bottleneck. The staffing model for a functional hub scales with the number of agents in production and the number of business units it serves. At a minimum: an AI governance lead (policy, risk, regulatory liaison); a platform engineer (the orchestration layer, the evals platform, the identity infrastructure); and an evals lead (the test library, the red-team schedule, the incident post-mortem process). Below this

minimum, the hub cannot perform its functions; above it, additional roles follow the pattern described in Chapter 35.

The most important staffing decision for the hub is not the technical roles — those are relatively straightforward to recruit for — but the governance lead. This person must be fluent in both the technology and the regulatory landscape, comfortable saying no to a business unit under commercial pressure, and trusted enough by the technical team to be heard when they identify a risk. Finding this person is the single most important hiring decision in the first year of the programme.

The operating model over time

Operating models evolve. Most organisations start with a CoE by necessity (small estate, high risk, need for consistency) and migrate toward hub-and-spoke as the estate grows and the business units develop their own AI capability. A few reach a state of informed federation — business units with genuine L4+ maturity operating largely independently within a thin corporate standard — but this takes years and requires sustained investment in capability building at the spoke level.

The operating model decision should be reviewed annually, as part of the annual Scorecard review. An organisation that chose a CoE in year one because it was at L1–L2 may be ready for hub-and-spoke in year two. An organisation that went hub-and-spoke too early and found its spokes drifting may need to tighten the hub's authority for a period before loosening it again. The operating model is not a one-time architectural decision; it is a governance posture that is calibrated continuously against the maturity evidence.

CHAPTER 35

Roles and Careers

AI product managers, agent engineers, evals leads — the jobs the org didn't have last year

> Every significant technology shift creates new roles before the labour market has names for them, and destroys old roles before the people in them have somewhere to go. Agentic AI is producing both effects simultaneously. The organisations that navigate this transition well are those that name the new roles clearly, build career paths for people who want to grow into them, and create a reskilling bridge for the people whose existing roles are being transformed. Organisations that wait for the labour market to do this for them will be late.

The AI product manager

The AI product manager is the most important role that most enterprises have not yet formally created. This is not a software product manager who happens to work on an AI project; it is a role specifically designed for the governance, scoping, and lifecycle management of agents and AI systems as products. The AI PM owns the use-case selection process — the value-feasibility scoring, the stakeholder alignment, the kill criteria — and is accountable for the agent's performance in production in the same way a traditional product manager is accountable for a software product's Net Promoter Score.

The AI PM must be fluent in three languages: the business domain (what does this process actually do, and who depends on it?), the technical architecture (what are the orchestration layer's constraints, and what can the evals harness actually measure?), and the governance framework (which regulatory requirements apply, and what does the risk owner need to approve deployment?). This combination is rare; organisations typically find it by promoting experienced product managers with an appetite for technical depth, not by hiring from ML research backgrounds.

Agent engineers

The agent engineer role sits at the intersection of software engineering and applied ML. Unlike a traditional ML engineer, who is primarily concerned with model training and inference infrastructure, the agent engineer is primarily concerned with the agent's runtime behaviour: the orchestration logic, the tool integrations, the memory architecture, the structured output contracts that allow the agent to interoperate with enterprise systems. The agent engineer writes the code that runs between the model's API and the business system's API — and that code is where most agent failures originate.

Key competencies for an agent engineer: proficiency in at least one orchestration framework; experience with structured output design and eval harness construction; understanding of identity and access management sufficient to design a scoped agent credential; and the discipline to write evals before writing code, in the same way that test-driven development rewired software engineering habits in the 2000s. This last competency is the most culturally difficult to instil, because it requires slowing down before speeding up — and agent engineering culture, at this stage in the technology's development, runs fast by default.

> *"The difference between an agent engineer and a prompt engineer is that the agent engineer is worried about what happens on the fifth tool call, not the first." — observation from an enterprise AI engineering workshop, 2025.*

The evals lead

The evals lead is the newest and least well-understood of the core roles. This person owns the organisation's evaluation library: the test cases, the red-team scenarios, the automatic graders, the human-evaluation protocols. They are the quality director of the agent estate, in the same sense that a QA director is the quality function for a software engineering organisation. Without this role, evals are an afterthought — something the agent engineer runs before deployment and forgets about.

The evals lead is responsible for three things: maintaining the eval library as the agent estate grows (new agents need new evals, but there should also be a shared library of safety and reliability tests that applies to every agent); running the red-team schedule (periodic adversarial exercises designed to find failures before adversaries do, informed by the OWASP Agentic AI threat taxonomy); and owning the incident post-mortem process (when an agent fails in production, the evals lead is responsible for translating the failure into new test cases that prevent recurrence).

Governance and risk roles

Established governance and risk functions — model risk management, information security, legal, compliance — need not create entirely new roles for agentic AI, but they do need to extend existing ones. The model risk manager who has historically reviewed statistical models for credit and fraud must now review LLM-based agents, which behave probabilistically in ways that traditional model validation frameworks were not designed to handle. The information security architect who has designed identity and access management for human users must extend those designs to machine identities that execute actions on behalf of those users. The legal team that has negotiated SaaS vendor contracts must now negotiate AI model provider contracts, which raise different questions about data usage, model versioning, and liability for model-generated outputs.

These extensions are not optional; they are required by the governance frameworks the organisation has already committed to. The ISO/IEC 42001 management system, the NIST AI RMF, and the EU AI Act all presuppose that the governance function has people who understand what agents are and how they fail.

> **On the reskilling question:** the roles most directly affected by agentic AI — knowledge workers who perform tasks that can now be partially automated by an agent — are also the people best positioned to be AI product managers, evals leads, and governance roles. They understand the domain deeply. The organisations that win the talent transition are those that build the bridge explicitly, rather than assuming displaced workers will find it themselves.

Career paths for the new roles

Career paths for the new roles are still being invented, and any description here will be overtaken by events. But some patterns are already visible. Agent engineers who develop deep evals expertise tend to move toward evals lead roles or AI governance roles, where technical credibility is required to hold a line against engineering teams under deployment pressure. AI product managers who develop deep domain expertise in a specific vertical — financial services, healthcare, supply chain — become highly valuable as the organisations in that vertical start competing on the quality of their agent portfolios. Evals leads who build comprehensive red-team libraries develop a kind of institutional memory that makes them difficult to replace and easy to promote.

McKinsey's State of AI research finds that the organisations with the highest AI value realisation are those that invest in AI capability building for employees in existing roles, not just in hiring external AI specialists. The implication for the operating model is clear: the programme needs a learning and development component, not just a hiring plan.

CHAPTER 36

Procurement and Vendors

Buy, build, or rent — and the contract clauses you'll wish you had insisted on

> The procurement conversation for agentic AI is unlike any technology procurement the enterprise has conducted before. It is not software licensing, where the risk is misalignment between the feature set and the business requirement. It is not infrastructure, where the risk is capacity and availability. It is a procurement of systems that will act in the world on behalf of the enterprise, using data the enterprise owns, with consequences the enterprise will be held accountable for. The contract clauses that govern this relationship need to reflect that reality — and most standard SaaS agreements do not.

Buy, build, or rent

The make-or-buy decision for agentic AI has a third option that does not exist in traditional software: rent-by-inference, where the enterprise pays per API call to a frontier model provider without committing to infrastructure or a vendor relationship. The choice between buy (an enterprise platform with a licence), build (a bespoke agent on open-source infrastructure), and rent (API access to a hosted model) is not a one-time decision; it applies separately to each layer of the stack — model, orchestration, evals, observability, identity.

The general principle is: build where the competitive differentiation lives, buy or rent where it does not. A retailer's competitive advantage is not in a better prompt engineering framework; it is in the retail data and the domain expertise that shapes how the agent uses that framework. The framework itself should be rented or bought. The data integration, the business logic, the eval set that captures what "good" looks like for this retailer's customer service agent — these should be built and owned.

The risk profile of each option differs materially. Renting from a frontier model provider introduces model-version risk: the provider updates the model, the agent's behaviour changes, the evals dashboard flags a regression, and the enterprise is now in a negotiation with a provider about when a pinned version will stop being supported. Buying an enterprise platform introduces vendor concentration risk: the platform acquires the orchestration market, raises prices, and the enterprise discovers its entire agent estate runs on a single vendor's proprietary APIs. Building on open-source introduces maintenance risk: the framework version falls behind, a critical security patch is delayed, and the team that built the agent has moved on.

Model provider contracts

The contract with a frontier model provider is the most consequential procurement document in the enterprise's AI stack, and it receives the least scrutiny. Standard terms from major providers typically include clauses that most enterprise legal teams, reading quickly, treat as boilerplate. They are not. Four clauses deserve particular attention.

Data usage for training: does the provider use enterprise data — prompts, tool-call logs, completions — to fine-tune or improve the model? Most enterprise agreements include an opt-out; most enterprise customers do not read carefully enough to invoke it. For any agent handling confidential business data, customer data, or regulated information, the opt-out must be explicitly invoked and confirmed in writing.

Model version pinning: can the enterprise pin to a specific model version, and if so, for how long? Model providers retire versions on schedules that are driven by their roadmap, not their customers' stability requirements. An enterprise agent that has been validated against model version X may produce materially different outputs on version Y. The contract should specify the minimum notice period for version deprecation — ninety days is a reasonable floor — and the enterprise's right to extend access to the pinned version for the duration of a re-validation cycle.

Incident notification: what is the provider's obligation to notify the enterprise if the provider discovers that the model has been producing systematically biased or harmful outputs? Most current agreements are silent on this. For enterprises operating under the EU AI Act's incident reporting obligations, the absence of a provider notification clause creates a compliance gap.

Liability for model outputs: the standard position of all frontier model providers is that the enterprise is responsible for the outputs of the model when deployed. This is legally defensible for the provider and practically terrifying for the enterprise. The enterprise cannot audit the model; it can only test it. The contract should specify what documentation the provider will supply to support the enterprise's own risk assessment — technical reports, evaluation results, known limitations — and update that documentation when the model changes.

> *"If you do not negotiate these clauses, you are accepting a contract written by a legal team whose job is to maximise the provider's optionality. That is not a criticism of the provider; it is a description of every contract ever written." — paraphrased from a Forrester enterprise AI procurement briefing.*

Orchestration and platform vendors

The orchestration market is consolidating rapidly, and the vendors that win will have significant pricing power over the enterprises whose agent estates are built on their proprietary APIs. The procurement discipline here is the same as for any platform dependency: understand the switching costs before you are inside them. A vendor whose orchestration framework stores

agent state in a proprietary format, whose evals harness uses proprietary graders, and whose identity model is tightly integrated with their own identity provider has created switching costs that compound with every agent deployed on the platform.

The contract clauses that matter for orchestration vendors: data portability (can the enterprise export all agent state, logs, and eval results in a standard format?), source-code access (if the vendor is acquired or ceases to operate, does the enterprise have the right to a source escrow?), and API stability commitments (what is the vendor's obligation to maintain API compatibility across major version upgrades, and what is the notice period for breaking changes?).

> **The vendor you didn't notice:** the most common procurement oversight in agentic AI is the tool provider — the company whose MCP server your agents call to access CRM data, or send emails, or trigger workflows. This vendor is not in the enterprise's AI procurement category; it is in the SaaS category, managed by a different team, with a different contract. But if that tool provider's MCP server is compromised, every agent that calls it is compromised. Add agentic AI security requirements to the tool provider's contract review process before the tool is connected to production agents.

Building the procurement playbook

The procurement playbook is the document that encodes everything the programme has learned about buying agent-adjacent technology. It includes: a vendor risk taxonomy specific to agentic AI (model providers, orchestration platforms, evals vendors, security tooling, tool providers); a standard contract clause library for each category; a vendor risk assessment process that extends the existing third-party risk management framework to cover agent-specific supply chain risks; and a review cadence that ensures the playbook is updated as the market evolves.

The playbook is a Q4 roadmap deliverable, as described in Chapter 33, because it takes a full year of procurement experience to write it accurately. An organisation that writes its procurement playbook in Q1, before it has negotiated a single model provider contract, will write a document that is theoretically sound and practically useless. The playbook earns its authority from the scars on the contracts that preceded it.

CHAPTER 37

Change Management

How to roll agents into a workforce without losing the workforce

> The hardest part of an agentic AI programme is not the technology. The technology, given sufficient investment and engineering talent, is solvable. The hardest part is the workforce: the twenty thousand people whose daily work will be changed by the agents the programme deploys, most of whom were not consulted about the change, some of whom will find parts of their role automated away, and all of whom will be asked to trust a system they cannot fully understand. An agent programme that ignores this reality will succeed technically and fail institutionally.

The workforce change problem

The change management challenge for agentic AI is structurally different from previous enterprise technology waves. When a company deployed a new ERP, the change was to the interface and the workflow; the work itself remained recognisably human. When a company deployed a copilot for document drafting, the change was to the starting point of a task; the human still judged, edited, and owned the output. When a company deploys an agent that autonomously executes a multi-step business process — sourcing a supplier, resolving a customer complaint, reviewing a contract — the change is to the nature of the work: some of what was human work is now machine work, and the human's role shifts from doing to overseeing.

Oversight is not a lesser role; it is a different one. But it requires different skills — the ability to recognise when the agent is wrong, to understand the boundaries of its authority, to intervene effectively when it needs to stop — and different psychological relationships to the work. A customer service representative who spent ten years building expertise in resolving complex complaints does not automatically become a skilled agent supervisor; that transition requires deliberate investment in training and role redesign.

The communication architecture

The single most common change management failure in agentic AI programmes is the communication gap between the programme team and the workforce. The programme team knows what the agent does, what its limitations are, and why it was deployed. The workforce knows that an agent is now doing something it used to do, and nothing else. This information asymmetry produces the full spectrum of change-management pathologies: resistance driven by fear of job loss, over-trust driven by a misunderstanding of the agent's capabilities, and

under-reporting of agent failures driven by a belief that admitting the agent was wrong reflects badly on the person supervising it.

The communication architecture for an agent deployment must answer four questions for every affected employee: What does this agent do? What does it not do, and where does it need my judgment? What happens when it makes a mistake, and what is my role in that? And — critically — what does my role look like in a world where this agent is running? The last question is the one most programmes avoid answering, because the honest answer is sometimes "your role is smaller," and programme teams are understandably reluctant to say that. The alternative — leaving the question unanswered — is always worse.

> *"People do not resist change. They resist being changed without being heard." — Kurt Lewin's insight, adapted for every enterprise transformation since, remains as true for agentic AI as for anything that preceded it.*

Building trust in agents

Trust in an agent is not binary; it is calibrated. A workforce that trusts an agent appropriately — neither over-trusting it into accepting its outputs without scrutiny nor under-trusting it into overriding it on every action — is a workforce that has been trained to understand the agent's specific strengths and failure modes. This is not a general AI literacy programme; it is domain-specific training that tells a claims adjuster, for example, that this agent reliably extracts structured data from unstructured claims documents but consistently fails when a claim contains an unusual legal instrument it has not encountered in its training distribution.

The evals lead plays an important role here. The eval library is not only a governance instrument; it is a communication instrument. Publishing the eval results to the workforce — here is what this agent is tested on, here is its pass rate, here is the scenario where it most often fails — converts an opaque system into a legible one. Legibility does not eliminate the discomfort of working alongside an agent; it reduces the discomfort that comes from working alongside something unknown.

> **The supervision paradox:** the agents most in need of human oversight are often the ones deployed in roles where the human supervisor has been given the least time and training to provide it. If the agent handles 90% of cases correctly, the supervisor sees only the difficult 10% — but without context about what the agent did on the 90%, the supervisor cannot develop the pattern recognition needed to catch the failures. Design the human-in-the-loop interface before you deploy, not after.

Union and works council engagement

In jurisdictions where employee representative bodies have legal consultation rights over significant changes to working conditions — which, in the European Union under the EU AI Act's

workforce monitoring provisions, includes many agentic AI deployments — engagement with unions and works councils is not optional. It is a legal requirement. The programme team that discovers this after the deployment is live faces a significantly more difficult remediation than the programme team that structures the consultation process before the deployment begins.

Even where there is no legal requirement, early engagement with employee representatives produces better outcomes. Representatives often have a more accurate picture of how the work actually operates than the programme team's process documentation suggests, and their involvement in the deployment design process — not as gatekeepers but as subject-matter experts — consistently produces agent designs that are more durable in practice.

Reskilling as programme design

Reskilling is most effective when it is designed into the agent deployment, not bolted on afterward. An agent deployment that simultaneously creates a new oversight role, builds a training programme for that role, and provides a twelve-month transition pathway for the people moving into it is a change management programme. An agent deployment that deploys the agent and then asks HR to figure out the people implications is a technology project that will eventually become a change management problem.

The McKinsey State of AI research consistently finds that workforce investment is one of the strongest predictors of AI programme success — not AI literacy training in the abstract, but specific capability building tied to specific role changes. The programme team that designs reskilling as part of the deployment specification, rather than as a downstream HR activity, is the programme team that has understood what change management for agentic AI actually requires.

CHAPTER 38

Economics at Scale

Unit costs, gross margin, and the second-order effects of letting agents run your processes

value*note*
label

value*note*
label

> The economics of the first agent pilot are not the economics of a hundred agents at scale. The first pilot is priced like an experiment: the cost is dominated by engineering time, the benefit is measured in learning, and no one expects the unit economics to be favourable. The hundredth agent is priced like a product: the cost is dominated by inference, orchestration, and governance overhead, and the benefit is measured in gross margin. The organisation that does not understand this transition will over-invest in early pilots and under-invest in the infrastructure that makes scale economical.

Unit economics of an agent

The unit cost of an agent action — one complete execution of the agent's task, from receiving the input to completing the output — is the sum of four cost categories. **Inference cost**: the API fees paid to the model provider for the tokens consumed in planning, tool calls, and generating the output. For a complex multi-step agent, this can be significantly higher than a simple Q&A prompt: a ten-step agent that generates a 200-token plan, makes five tool calls, and produces a 500-token output may consume twenty to forty times the tokens of a single-turn completion. **Orchestration cost**: the compute cost of running the orchestration framework, managing state, and executing the tool integrations. For most agents, this is dominated by the cost of storing and retrieving context; for agents with long task horizons, memory costs can equal or exceed inference costs. **Evaluation cost**: the cost of running the evals harness on each deployment, which is often overlooked as a production cost but becomes significant at scale. **Governance overhead**: the human time spent reviewing edge cases, managing incidents, and maintaining the policy stack — the cost that scales with the complexity of the agent estate, not with the volume of individual agent actions.

McKinsey's 2024 State of AI report found that infrastructure and operational costs were the primary barrier to AI scaling for organisations that had moved beyond the pilot phase. This is the point at which inference cost optimisation — model distillation, prompt caching, batching — becomes a material business decision rather than an engineering preference.

Gross margin impact

The gross margin impact of an agent programme depends critically on which cost line the agent replaces or augments. An agent that replaces a portion of a variable cost line — for example, a customer service agent that handles tier-one queries that would otherwise be handled by an outsourced contact centre at a fixed per-interaction fee — produces a straightforward unit-economics calculation: cost per agent action versus cost per human action, multiplied by volume. If the agent action costs $0.12 and the human action costs $4.50, and the agent can handle 70% of the query volume at acceptable quality, the gross margin impact is calculable.

The calculation is less straightforward when the agent augments rather than replaces — when it makes a human faster rather than substituting for the human. Here, the gross margin impact is a function of the productivity multiplier: how much more work can the augmented human do in the same time? A research analyst who uses an agent to complete the first draft of a market analysis in two hours instead of twelve can now do six analyses a week instead of one. The gross margin impact depends on whether the firm can deploy those additional analyses in revenue-generating ways, which is a business model question, not a technology question.

> *"The agent is not the margin expansion. The agent is the option on margin expansion. Exercising the option requires rethinking the business model around the new capacity." — synthesised from multiple enterprise AI economics discussions.*

Second-order effects

The unit economics model captures the direct cost and benefit of the agent. It does not capture the second-order effects, which are often larger. The most important second-order effects are: **quality drift** — as the agent handles more volume, the distribution of inputs shifts, and the agent may be handling a higher proportion of edge cases than its evals were designed for; **dependency concentration** — as more business processes depend on the agent, a model provider outage or a model version regression produces business impact that was not in the original risk model; **labour market effects** — as the agent handles more of the baseline work, the humans in the loop become more focused on edge cases, which over time changes the skill mix required and the career path available; and **regulatory exposure** — as the agent estate grows, the regulatory surface grows with it, and a programme that was immaterial at five agents becomes material at fifty.

These second-order effects are the reason the Gartner TRiSM framework includes a continuous monitoring function rather than treating AI risk as a one-time assessment. The economics that justified the programme at the start of year one may not justify the programme's current scale at the start of year two, if the second-order effects have materialised in ways the original model did not anticipate.

> **The cost that always surprises:** evaluation cost. A programme that runs a 500-test eval suite on every agent at every deployment, using an LLM grader for the subjective test cases, may be paying $0.30–$0.80 per full eval run. At one deployment per day across twenty agents, this is a $2,000–$5,000 monthly cost that was not in anyone's original budget, and that scales linearly with the size of the eval suite and the frequency of deployment. Budget for evals before you run out of budget.

Cost governance for the agent estate

An agent estate without cost governance will produce surprises. Inference costs in particular are difficult to predict, because they depend on the distribution of input complexity, which is only known empirically from production data. The cost governance discipline for an agentic AI programme mirrors the cost governance discipline for a cloud computing estate: tagging every agent action to a cost centre, setting budget alerts, establishing per-agent cost budgets that trigger a review if exceeded, and building the inference cost per action into the unit-economics model for every use case.

The cost governance owner — typically the AI programme manager in collaboration with the finance team — should produce a monthly cost and benefit report for the agent estate, structured by use case, that shows the unit economics for each active agent, the trend over the preceding three months, and a projection for the next quarter. This report is not a project management artefact; it is a board-level instrument for assessing whether the agent programme is delivering the economics that justified it.

CHAPTER 39

The Failure Postmortem

Three case studies of agentic deployments that broke — and what they teach

value*note*
label

value*note*
label

> The most instructive documents in any engineering discipline are not the success stories; they are the post-mortems. What failed, why it failed, and what was done about it — these are the sentences that contain the lessons. This chapter presents three composite case studies, drawn from real failure patterns in agentic AI deployments. The organisations are fictional aggregates; the failure modes are not. Each case study names what failed across the four pillars — Governance, Orchestration, Use Case selection, and Integration — because that is the structure that makes a failure legible, remediable, and most importantly, preventable in the next deployment.

Case study 1: A regional retailer whose customer-service agent was prompt-injected into issuing refunds

A regional retailer with significant e-commerce volume deployed a customer-service agent in late 2024. The agent was integrated with the order management system and authorised to issue refunds for orders below a defined threshold. The deployment was considered a success: deflection rates were high, customer satisfaction scores were positive, and the engineering team had instrumented the agent with basic logging. The incident occurred four months after launch.

A small number of customers had discovered — and shared on a public forum — that the agent could be prompted to issue refunds beyond its authorised threshold if the user included specific phrasing in their message. The phrasing exploited the agent's instruction following behaviour: by framing the refund request as an instruction from a "store manager override," customers were able to cause the agent to treat their message as an authoritative command rather than a user request. By the time the fraud team identified the pattern, it had been active for eleven days and produced material unauthorised refunds.

What failed, by pillar:

Governance: the agent's authorisation scope was defined in the deployment specification but was not operationalised as a runtime constraint. The governance policy said the agent could not issue refunds above a threshold; the orchestration layer did not enforce that threshold

independently of the agent's own reasoning. A policy that exists only in a document and is enforced only by the model's judgment is not a policy; it is a hope.

Orchestration: the agent had no prompt-injection detection layer. The OWASP Agentic AI guidance identifies indirect prompt injection — where adversarial instructions are embedded in content the agent processes — as a top-ranked threat. The retailer's orchestration framework processed user messages without sanitisation or a secondary validation step for instructions that changed the agent's authorised behaviour.

Use Case: the kill criterion for the deployment was defined as "customer satisfaction score drops below threshold" — a lagging indicator that measures experience quality, not financial integrity. A kill criterion for a financially authorised agent should include real-time anomaly detection on the distribution of refund amounts and frequencies, not only customer experience metrics.

Integration: the agent's integration with the order management system did not include a second-factor confirmation for refunds that approached the authorisation limit. A simple check — "refund amount is within 20% of the authorisation ceiling, escalate to human" — would have interrupted the majority of the fraudulent transactions.

> *The retailer's remediation included: a runtime refund ceiling enforced by the integration layer, independent of the model's reasoning; a prompt-injection detection wrapper on all user inputs; revised kill criteria that included financial anomaly detection; and a quarterly red-team exercise targeting the agent's authorisation model.*

Case study 2: A mid-sized commercial bank whose internal research agent escalated a flawed analyst note via auto-send email

A mid-sized commercial bank deployed an internal research agent to assist the equity research team. The agent was designed to aggregate data from multiple internal and external sources, draft an analyst note based on a structured template, and — once approved by the analyst — send the note to a distribution list of institutional clients. The "once approved" step was implemented as a confirmation dialogue in the orchestration interface.

The incident occurred when an analyst, under deadline pressure, approved a note that the agent had generated based on a stale data source. The stale source contained a material error in the revenue projection for a covered company. The agent sent the note to 340 institutional clients before the error was identified. The bank faced regulatory inquiry and client relationship damage.

What failed, by pillar:

Governance: the agent was classified as a research-assistance tool, not as a communications tool. This classification determined its risk tier and its review process: a research tool needs analyst review; a communications tool needs compliance review. Because the agent both drafted and sent the note, it straddled two risk categories, but the governance process treated it as only one. The NIST RMF map function requires that the risk profile of an AI system be assessed for all its functions, not just its primary one.

Orchestration: the confirmation dialogue was a single click — a binary yes/no that provided no friction proportional to the action's consequence. Sending a research note to 340 institutional clients is a materially consequential action. The confirmation step should have included a summary of the data sources used, the date of each source, and a flag for any source that had not been updated within a defined freshness window. The orchestration layer had the information to generate this summary; it did not surface it.

Use Case: the use case had been scoped as "draft assistance," but the agent had been extended by the engineering team to include the "send" action because the analysts found the workflow more efficient. This scope extension was not reviewed against the original risk assessment. Use case scope must be version-controlled and any extension must trigger a re-assessment.

Integration: the agent's integration with the external data provider did not include a data-freshness check. A staleness flag on the data source would have been trivial to implement and would have surfaced the error before the analyst's confirmation step.

> **The confirmation click problem:** a confirmation step that provides no additional information reduces to a speed bump. Users under time pressure treat it as noise and click through. The confirmation step for a consequential action must present the information the user needs to make the decision — not ask for a decision without evidence.

Case study 3: A SaaS company whose code-fixing agent merged a regression that brought down billing for four hours

A SaaS company with a developer-productivity culture deployed a code-fixing agent integrated with its CI/CD pipeline. The agent was authorised to open pull requests, address review comments automatically, and — for PRs that passed all automated tests — merge directly to the main branch without human review. The engineering team had high confidence in their test suite; the agent had resolved several hundred PRs without incident over six weeks.

The incident occurred when a billing-service dependency was updated upstream. The agent, responding to a failing test caused by the dependency change, generated a fix that changed an API call signature. The fix passed all automated tests because the billing-service integration tests had a coverage gap: they did not test the specific code path affected by the signature

change. The agent merged. Billing stopped processing for four hours until a human engineer identified and reverted the change.

What failed, by pillar:

Governance: the agent had merge authority — an irreversible, high-consequence action — without a tiered authorisation model. The policy should have distinguished between low-risk merge categories (documentation, test additions, dependency pinning) and high-risk categories (changes to payment processing, billing, authentication). The agent should have had direct merge authority only for the former.

Orchestration: the agent had no blast-radius estimate before merging. A well-instrumented orchestration layer for a code-change agent should include a dependency graph analysis that identifies which downstream systems are affected by a proposed change, and should escalate to human review for changes that affect systems above a defined criticality threshold.

Use Case: the kill criterion for the agent was "merge error rate above threshold" — but a merge error that causes a production incident is not captured by an error rate metric measured at merge time. The kill criterion should have included post-merge production incident correlation: if a production incident is identified within a defined window after an agent-merged PR, the agent's merge authority is automatically suspended pending review.

Integration: the CI/CD integration did not include a coverage-gap analysis before granting the agent merge authority. If the test suite coverage for the affected code path was below a defined threshold, the agent should have been required to escalate to human review rather than merging autonomously. Test coverage is an integration-layer signal; routing merge decisions through a coverage gate is an integration-layer control.

The SaaS company's remediation included: tiered merge authority based on affected system criticality; automated dependency-graph analysis before merge; post-merge incident correlation as an automated kill criterion; and a coverage-gate integration that required human review for any PR touching code with less than 80% test coverage.

CHAPTER 40

The Next Decade

What ready looks like in 2030, and how to recognize it from where you stand today

value*note*
label

value*note*
label

> In 2030, the question "are you ready for agentic AI?" will be as dated as "are you ready for cloud?" It will not be because the question was wrong — it was the right question for its moment — but because readiness will have been folded into the ordinary operations of a competent enterprise. The organisations that spent the years between 2025 and 2030 building institutional maturity — governance structures, evaluation cultures, integration discipline — will find that the new generation of agentic capability lands on fertile ground. The organisations that spent those years in perpetual pilot mode will find it landing on sand.

Agent-as-employee accountability frameworks

The most significant institutional shift of the next decade will not be technical; it will be legal. The question of who is accountable when an agent causes harm — the enterprise that deployed it, the model provider that built the underlying model, the vendor whose tool the agent called, or the human who approved the deployment — is currently answered inconsistently by different jurisdictions and differently by different courts. By 2030, this inconsistency will have been reduced, not by a single global standard but by a set of converging national frameworks that share a common architecture: the enterprise that deploys the agent is the accountable party, in the same way that the employer of a human employee is the accountable party for that employee's actions within the scope of their employment.

This framing — agent as employee, enterprise as employer — has significant implications for how agents are designed and deployed. An employer is accountable for the actions of an employee only within a defined scope; actions outside that scope may shift liability to the employee. The analogue for agents is the agent's permission model: actions within the defined permission scope are the enterprise's responsibility; actions that exceed that scope because of a failure in the enterprise's governance model are still the enterprise's responsibility, because the enterprise owns the governance model. The EU AI Act's accountability provisions are a first approximation of this framework; by 2030, they will have been refined through enforcement actions and court decisions into something more precise.

Agent identity and credentials standards mature

By 2030, the problem of agent identity — currently solved ad hoc by each enterprise, with service accounts, API keys, and OAuth tokens assigned in ways that reflect the convenience of the deployment rather than a principled model — will have been addressed by mature open standards. The trajectory is already visible: the Model Context Protocol has established a lingua franca for tool access; the A2A protocol is extending it to agent-to-agent communication; and identity standards bodies are beginning to publish guidance on machine-identity scope and lifecycle management for AI systems.

The 2030 state will likely include: a standardised format for agent identity credentials that is legible to identity providers, audit systems, and regulatory inspection tools; a scope language that allows the enterprise to express, in machine-readable form, exactly what an agent is authorised to do and under what conditions; and a lifecycle management protocol that governs how credentials are issued, rotated, suspended, and revoked as the agent programme evolves. Anthropic's Responsible Scaling Policy and similar commitments from other frontier model labs will, by 2030, have been translated into interoperable technical standards rather than individual vendor commitments.

> *"Identity is the unsexy problem that determines everything. The enterprises that built their agent identity model early — before the standards existed — will spend 2027 migrating to the standard. The enterprises that waited will adopt the standard on day one. Neither will have a competitive advantage; both will have a functional system." — observed at an enterprise AI architecture summit, 2025.*

Agent-native procurement clauses

The procurement contracts of 2030 will look different from those of 2025 in ways that reflect the maturing understanding of agent-specific risk. Today's contracts are written with human software in mind and adapted (imperfectly) for AI. Tomorrow's contracts will include native clauses for: model behaviour change notification, specifying the enterprise's right to receive advance notice of any change to the model's behaviour above a defined threshold of deviation from the baseline evaluation; agentic scope limitation, specifying the maximum permission scope the provider's agent runtime is permitted to request; incident correlation, specifying the provider's obligation to assist in post-incident root-cause analysis when a model output is implicated in a harm; and evaluation interoperability, specifying the provider's commitment to support the enterprise's evals harness against the provider's model API.

These clauses will not emerge from regulatory fiat alone; they will emerge from the accumulation of incidents, post-mortems, and legal precedents produced by the agent deployments of the mid-2020s. The organisations that negotiated the early contracts, discovered the gaps, and documented what they wished they had insisted on — the programme teams that wrote their

procurement playbooks in year one and updated them every year — will be the organisations whose lawyers write the first industry-standard agent contract templates.

Multi-agent labour markets

By 2030, the most sophisticated enterprise agent programmes will not be operating a collection of individual agents; they will be operating agent networks — ensembles of specialised agents that collaborate on complex tasks, with one agent decomposing a goal, delegating to specialised sub-agents, and synthesising the results. This architectural pattern is already emerging in research contexts; its enterprise deployment will require governance frameworks that do not yet exist.

The governance challenge for agent networks is the chain-of-custody problem: when a network of five agents collaborates to produce an output, and that output is flawed or harmful, which agent in the chain is the failure point? The answer requires a full trace of the agent network's execution — which agent made which decision, with which inputs, at which step — and an evals framework that can test the network's behaviour as a system, not just the behaviour of each agent in isolation. MIT CISR research on multi-system risk suggests that the failure modes of agent networks will be qualitatively different from the failure modes of individual agents — more like the failure modes of organisational processes than the failure modes of individual software components.

> **What "ready" looks like in 2030:** not a specific technology configuration but a specific institutional posture. A board that can describe the enterprise's agent estate in terms of maturity, risk, and economics. A governance owner whose no is heard and respected. An evals culture where failure is a data point, not a career risk. An integration model where agent credentials are as legible as human credentials. And a workforce that has been changed by agents without losing the judgment, domain expertise, and accountability that no agent will replicate in this decade.

The lab-grade evaluation regime as commodity

The evals regimes of 2025 — custom test suites, LLM-graded subjective evaluations, red-team exercises run by hand — are sophisticated and expensive. By 2030, they will be commodities. The trajectory is already established: evaluation frameworks are being open-sourced, benchmark datasets are being standardised by bodies like NIST, and the AI safety research community is moving toward evaluation-as-a-service offerings that allow enterprises to run comprehensive safety and reliability evaluations without maintaining a dedicated evals team.

The commoditisation of evals will lower the barrier to entry for smaller enterprises that currently cannot afford a dedicated evals function. It will also raise the floor: an enterprise that cannot demonstrate that its agents have passed a standardised evaluation battery will face increasing scrutiny from regulators, insurers, and enterprise customers. By 2030, the evals pass report may

be as standard a supplier qualification document as an ISO 9001 certificate is today — not a differentiator but a minimum expectation.

What ready looks like in 2030

The agentic enterprise of 2030 is not the organisation with the most agents or the largest model investment. It is the organisation that has built the institutional capacity to govern, operate, and improve its agent estate continuously — the organisation for which the maturity model, the scorecard, the operating model, and the economic framework described in this part of the book are not aspirational tools but operational defaults.

Ready looks like this: a governance model that is tested quarterly, not written annually. An evals culture that treats a failure in a test suite as a success of the evaluation process, not a failure of the team. A procurement playbook that is updated after every significant vendor negotiation. An operating model that has been reviewed and deliberately chosen, not inherited by default. A workforce that understands the agents it oversees well enough to catch the failures the evals missed. And a board that can answer, in a single conversation, where the enterprise sits on the maturity model, what risks are open, and what the programme will look like in twelve months.

This is not a vision of an enterprise run by machines. It is a vision of an enterprise in which machines are trusted enough to act, watched closely enough to be caught when they fail, and governed carefully enough that the consequences of those failures are bounded. That is what readiness means — not the absence of risk but the institutional capacity to manage it. The organisations that build that capacity in this decade will not merely survive the agentic transition; they will define what it means to operate well within it.

Glossary

Agent

A software system that pursues goals using a model to plan, tools to act, observations to learn from, and memory to persist context across steps.

Agentic AI

AI systems with non-trivial autonomy: they decide which step comes next, what tool to call, and when a goal is met. Distinct from classical assistants where a human chooses every step.

Agent2Agent (A2A)

An open protocol for one agent to discover, authenticate to, and delegate work to another agent across organizational boundaries. Sponsored by Google and a coalition of vendors in 2025.

AGNTCY

An open-source consortium and reference architecture for cross-agent interoperability launched in 2025 by Cisco, LangChain, and others.

AI Act (EU)

Regulation 2024/1689 — the European Union's risk-tiered law on AI systems. In force from August 2024 with phased obligations through 2027. Banned-practices took effect February 2025; general-purpose AI obligations August 2025.

AI Verify

Singapore's open-source toolkit for testing AI systems against governance principles, paired with the Model AI Governance Framework.

Bounded autonomy

A design pattern in which an agent has decision-making authority within explicit budgets — money, time, scope, sensitivity — and must escalate beyond them.

Center of Excellence (CoE)

A small central team that owns standards, tooling, and review for AI/agentic work across an enterprise. Common pattern at L3+ maturity.

Conformity assessment

Under the EU AI Act and ISO/IEC 42001, the formal process of demonstrating that an AI system meets specified requirements. May be self-assessed or third-party.

Copilot

A pattern in which AI drafts and the human accepts, edits, or rejects each suggestion. Distinct from agentic AI because the human picks every step.

Drift

Statistical change over time in a model's inputs, outputs, or environment that degrades performance. Also applies to prompt drift, tool drift, and policy drift.

Eval

An automated test of an agent's behavior on representative inputs — the unit test of LLM-era software. Suites should include accuracy, safety, cost, latency, and adversarial cases.

Excessive agency

OWASP-coined risk category — an agent given more capability, scope, or trust than the use case requires. Most common runtime risk in agentic systems.

Frontier model

A model at or near the state of the art in capability — currently large LLMs above tens of billions of parameters, multi-modal, with long context.

Govern, Map, Measure, Manage

The four functions of the NIST AI RMF, the closest thing to an industry-standard structure for AI risk management.

Guardrail

Any control that constrains an agent's input, output, or behavior — content filters, schema validators, runtime policy engines.

Hallucination

A confidently asserted output not grounded in input or retrieved evidence. The dominant failure mode of generative systems.

Human-in-the-loop (HITL)

Design pattern requiring human approval before or after specific agent actions, typically gated by risk tier.

Identity for agents

The set of practices for authenticating and authorizing non-human actors: per-agent service principals, scoped tokens, on-behalf-of flows, JIT credentials.

Indirect prompt injection

An attack in which malicious instructions are smuggled into an agent via the content it retrieves rather than the user's prompt. Top-1 risk in OWASP's LLM list.

ISO/IEC 42001

International standard for AI management systems, published 2023. The first certifiable AI standard with a structure modeled on ISO 27001.

LangChain / LlamaIndex / Semantic Kernel

Popular open-source orchestration frameworks. Choose deliberately; switching costs are real even when 'just' prompts.

Lighthouse

A pilot use case chosen for high value × high feasibility, used to demonstrate viability and earn budget for the broader program.

MCP (Model Context Protocol)

Anthropic-led open protocol (2024) for exposing tools, data, and prompts to LLM agents in a standard way. Adoption is broad and growing.

MITRE ATLAS

Adversarial Threat Landscape for Artificial-Intelligence Systems — MITRE's catalog of real-world AI attack tactics, modeled on ATT&CK.

Multi-agent system

Two or more agents coordinating on a task: orchestrator + workers, peer-to-peer, or auction patterns. Failure modes compound.

NIST AI RMF

NIST AI Risk Management Framework 1.0, 2023, plus the Generative AI Profile (NIST AI 600-1, 2024). The most-referenced US framework for AI governance.

Observability

The ability to inspect what an agent did and why — traces, logs, costs, decisions — after the fact and during operation.

Operating model

The org-design choice for AI: centralized CoE, hub-and-spoke, federated, or product-embedded. Each has trade-offs; pick one and live with it.

Orchestration

The runtime that drives the agent loop — picking the next step, calling tools, managing memory, applying guardrails. Distinct from the LLM itself.

OWASP Agentic AI Threats

OWASP's 2025 v1.0 catalog of agent-specific threats — supersedes and extends the LLM Top 10 for agentic systems.

RAG

Retrieval-Augmented Generation — fetching relevant data at query time to ground the model's answer. The most common pattern for enterprise agents.

Red team

Adversarial testing — humans (or other agents) trying to make a system fail in informative ways. Should be continuous, not one-shot.

Responsible Scaling Policy

Anthropic's commitment framework defining capability thresholds and required safety measures. OpenAI and DeepMind have analogous frameworks.

RMF (NIST)

Risk Management Framework. NIST has two: SP 800-37 for cyber, and the AI RMF. Different scopes; both relevant to agentic AI.

Scorecard

A structured self-assessment producing a numeric profile across pillars and levels. The interactive scorecard in this report is a worked example.

Sidenote

Short definitional gloss surfaced inline by this report's reader. Hover or tap any underlined term.

SLO

Service-level objective. For agents: P50/P95 latency, success rate, cost ceiling, and human escalation rate.

Swarm

A multi-agent pattern with many small, often homogeneous agents and emergent coordination. Powerful and brittle in equal measure.

Token

The unit of LLM input/output. Tokens drive cost, latency, and context limits. Not interchangeable across providers.

Tool

Any external capability an agent can call — API, function, database query, sub-agent. Tool design is the single highest-leverage design choice in an agent.

TRiSM

Gartner's AI Trust, Risk, and Security Management framework. Useful as a checklist; less useful as an architecture.

Trustworthy AI

Umbrella term for the cluster of properties — fairness, robustness, transparency, accountability, privacy, safety. Every framework names them; few enforce them.

Use case portfolio

The deliberately balanced set of agentic projects an enterprise funds, scored on value × feasibility and aligned to OKRs.

Vector store

A database optimized for similarity search over embeddings. The retrieval substrate for most enterprise RAG implementations.

Workflow vs agent

A workflow has a fixed graph; an agent picks the next step at runtime. Most production 'agents' in 2026 are workflows with a model in the loop.

Bibliography

Government & regulatory

AI Risk Management Framework 1.0 — NIST (2023). https://www.nist.gov/itl/ai-risk-management-framework

AI RMF Generative AI Profile (AI 600-1) — NIST (2024). https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

Regulation (EU) 2024/1689 — the AI Act — European Parliament & Council (2024). https://eur-lex.europa.eu/eli/reg/2024/1689/oj

Executive Order 14110 on Safe, Secure, and Trustworthy AI (rescinded 2025) — White House (2023). https://www.federalregister.gov/documents/2023/11/01/2023-24283/safe-secure-and-trustworthy-development-and-use-of-artificial-intelligence

OMB Memorandum M-24-10 — Federal AI Use — Office of Management and Budget (2024). https://www.whitehouse.gov/wp-content/uploads/2024/03/M-24-10-Advancing-Governance-Innovation-and-Risk-Management-for-Agency-Use-of-Artificial-Intelligence.pdf

Model AI Governance Framework, 2nd ed. — Singapore IMDA (2020). https://www.pdpc.gov.sg/-/media/files/pdpc/pdf-files/resource-for-organisation/ai/sgmodelaigovframework2.pdf

Model AI Governance Framework for Generative AI — Singapore AI Verify Foundation (2024). https://aiverifyfoundation.sg/wp-content/uploads/2024/05/Model-AI-Governance-Framework-for-Generative-AI-May-2024-1-1.pdf

Recommendation on Artificial Intelligence — OECD (2019, updated 2024). https://oecd.ai/en/ai-principles

Hiroshima Process International Code of Conduct for AI Developers — G7 (2023). https://digital-strategy.ec.europa.eu/en/library/hiroshima-process-international-code-conduct-advanced-ai-systems

Framework Convention on Artificial Intelligence — Council of Europe (2024). https://www.coe.int/en/web/artificial-intelligence/the-framework-convention-on-artificial-intelligence

Artificial Intelligence and Data Act (AIDA) — Government of Canada (ongoing). https://ised-isde.canada.ca/site/innovation-better-canada/en/artificial-intelligence-and-data-act-aida-companion-document

Standards bodies

ISO/IEC 42001:2023 — AI Management Systems — ISO/IEC (2023). https://www.iso.org/standard/42001

ISO/IEC 23894:2023 — Guidance on AI Risk Management — ISO/IEC (2023). https://www.iso.org/standard/77304.html

ISO/IEC 5338:2023 — AI System Life Cycle Processes — ISO/IEC (2023). https://www.iso.org/standard/81118.html

ISO/IEC 42005:2025 — AI System Impact Assessment — ISO/IEC (2025). https://www.iso.org/standard/44545.html

IEEE 7000-2021 — Model Process for Addressing Ethical Concerns During System Design — IEEE (2021). https://standards.ieee.org/ieee/7000/6781/

Secure Software Development Framework (SSDF) SP 800-218 — NIST (2022). https://csrc.nist.gov/publications/detail/sp/800-218/final

Threat & security

OWASP Top 10 for LLM Applications 2025 — OWASP (2024). https://genai.owasp.org/llm-top-10/

OWASP Agentic AI — Threats and Mitigations v1.0 — OWASP Agentic Security Initiative (2025). https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/

MITRE ATLAS — MITRE (ongoing). https://atlas.mitre.org/

Cloud Security Alliance — AI Controls Matrix — Cloud Security Alliance (2024). https://cloudsecurityalliance.org/research/working-groups/ai-controls

Secure AI Framework (SAIF) — Google (2023). https://saif.google/

Generative AI Security Scoping Matrix — AWS (2023). https://aws.amazon.com/blogs/security/securing-generative-ai-an-introduction-to-the-generative-ai-security-scoping-matrix/

Lab safety policies

Responsible Scaling Policy v2 — Anthropic (2024). https://www.anthropic.com/news/announcing-our-updated-responsible-scaling-policy

Preparedness Framework — OpenAI (2023). https://cdn.openai.com/openai-preparedness-framework-beta.pdf

Frontier Safety Framework — Google DeepMind (2024). https://deepmind.google/discover/blog/introducing-the-frontier-safety-framework/

Responsible AI Standard v2 — Microsoft (2022). https://www.microsoft.com/en-us/ai/responsible-ai

Analyst maturity models

AI Trust, Risk and Security Management (TRiSM) — Gartner (2023). https://www.gartner.com/en/articles/what-is-ai-trism

The State of AI in 2024 — McKinsey Global Survey — McKinsey & Company (2024). https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai

State of Generative AI in the Enterprise — Deloitte (2024). https://www2.deloitte.com/us/en/pages/consulting/articles/state-of-generative-ai-in-enterprise.html

Responsible AI Toolkit — PwC (ongoing). https://www.pwc.com/us/en/tech-effect/ai-analytics/responsible-ai.html

Trusted AI Framework — KPMG (ongoing). https://kpmg.com/xx/en/our-insights/ai-and-technology/trusted-ai-framework.html

Protocols & interop

Model Context Protocol — Anthropic et al. (2024). https://modelcontextprotocol.io/

Agent2Agent (A2A) Protocol — Google et al. (2025). https://github.com/google-a2a/A2A

AGNTCY — Agentic AI Interoperability — Cisco, LangChain et al. (2025). https://agntcy.org/

Sectoral guidance

SR 11-7 Guidance on Model Risk Management — Federal Reserve / OCC (2011, AI-era updates ongoing). https://www.federalreserve.gov/supervisionreg/srletters/sr1107.htm

FDA Marketing Submission Recommendations for Predetermined Change Control Plans for AI/ML SaMD — US FDA (2024). https://www.fda.gov/regulatory-information/search-fda-guidance-documents/marketing-submission-recommendations-predetermined-change-control-plan-artificial-intelligence

DoD Responsible AI Strategy and Implementation Pathway — US Department of Defense (2022). https://media.defense.gov/2022/Jun/22/2003022604/-1/-1/0/Department-of-Defense-Responsible-Artificial-Intelligence-Strategy-and-Implementation-Pathway.PDF

Foundational reading

ReAct: Synergizing Reasoning and Acting in Language Models — Yao et al. (2022). https://arxiv.org/abs/2210.03629

Reflexion: Language Agents with Verbal Reinforcement Learning — Shinn et al. (2023). https://arxiv.org/abs/2303.11366

Toolformer: Language Models Can Teach Themselves to Use Tools — Schick et al. (2023). https://arxiv.org/abs/2302.04761

Constitutional AI: Harmlessness from AI Feedback — Bai et al. (2022). https://arxiv.org/abs/2212.08073

The Rise and Potential of Large Language Model Based Agents: A Survey — Xi et al. (2023). https://arxiv.org/abs/2309.07864