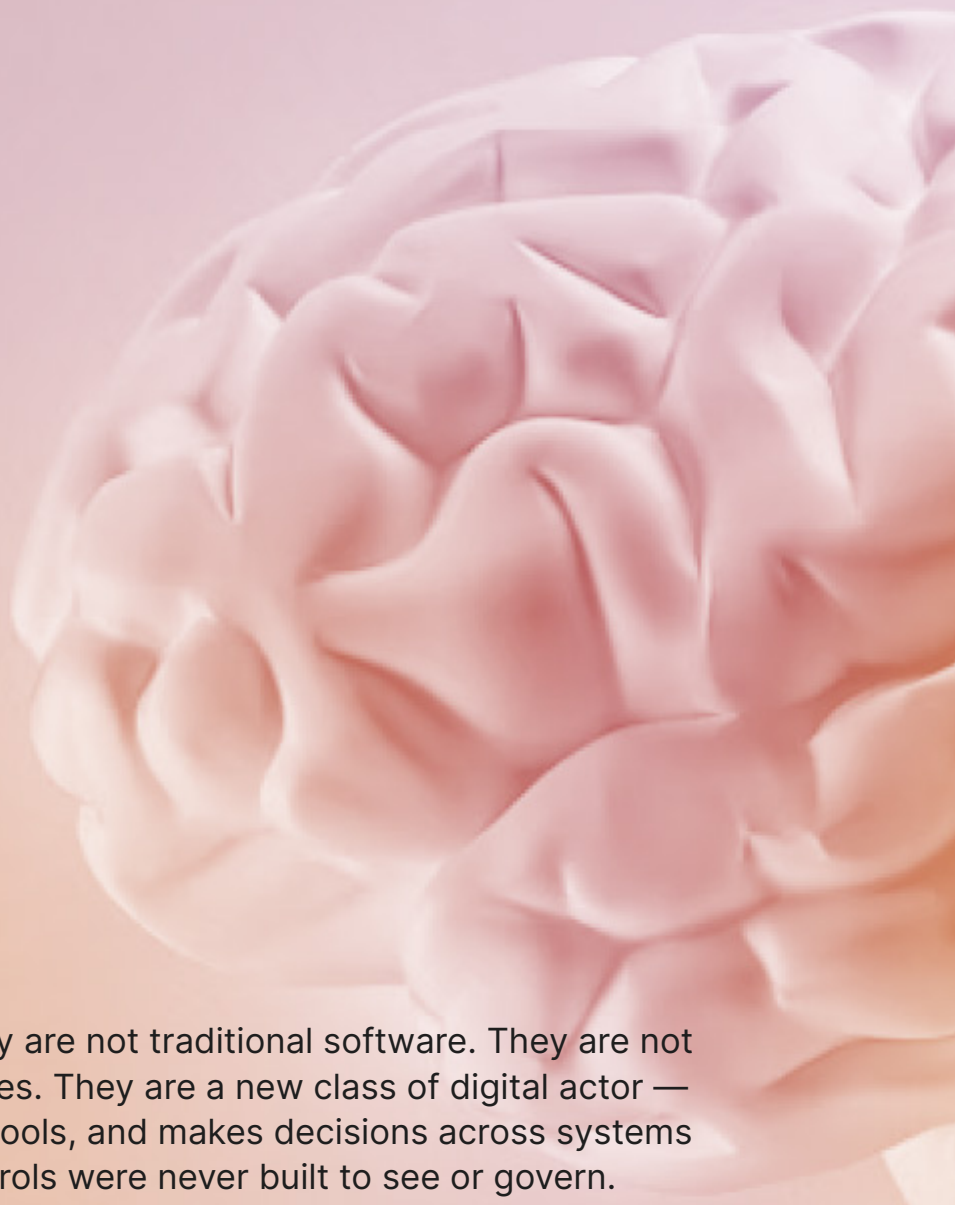
APRIL 2026

Top 10 AI Agent governance best practices

Go beyond identity to a broader maturity model of prompts, guardrails and behavior

Geordie

Table of Contents



Introduction

AI agents are not automation. They are not traditional software. They are not human users with unusual privileges. They are a new class of digital actor — one that interprets tasks, selects tools, and makes decisions across systems in ways that existing security controls were never built to see or govern.

The controls organisations rely on today — identity providers, Zero Trust architectures, access policies — remain essential. They establish the boundaries within which agents operate. They do not explain what agents do within those boundaries, how decisions unfold across a multi-step workflow, or when behaviour moves outside what the organisation intended.

That gap is where agent risk lives. And closing it requires governance practices that go beyond access. This is what those practices look like.

1

Know what your agents can reach — and what they actually do once they get there.

Access controls tell you which systems an agent can reach. They don't tell you how it behaves once it arrives. Both questions matter. Most current security tooling only answers the first one.

Most security leaders have made substantial progress in governing access. Identity providers, Zero Trust architectures, and least-privilege models now give organisations strong control over which systems AI agents can reach and what credentials they carry. These controls all assume boundaries for human users and choke points along the way where access can be controlled.

But an agent can hold appropriate permissions and still behave in ways that violate policy, extend data boundaries, or create operational risk. Identity does not explain what agents actually do once they arrive, how an agent will interpret a task or execute a sequence of actions.

This means that now we must look closer at behavior, which brings security teams to a governance problem that sits above identity and access management.

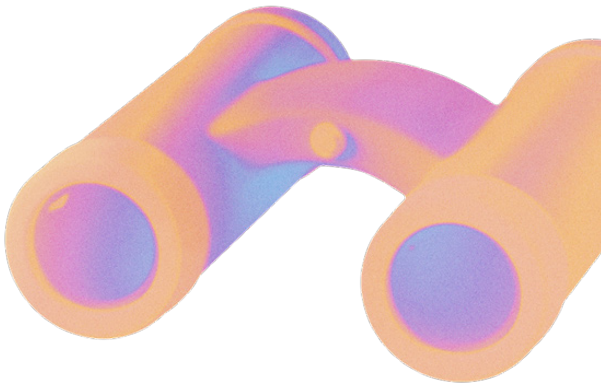
Two questions now matter:

1. What systems can the agent reach?

2. How does the agent behave once it reaches them?

Most current security tooling answers the first question.

The second requires different instrumentation and different practices.



2

Treat your agents like digital employees — not automation scripts.

Traditional automation executes predefined instructions. Agents interpret tasks, select tools and make decisions in real time. Their risk profile depends on how they interpret instructions and how they act when situations require judgement.

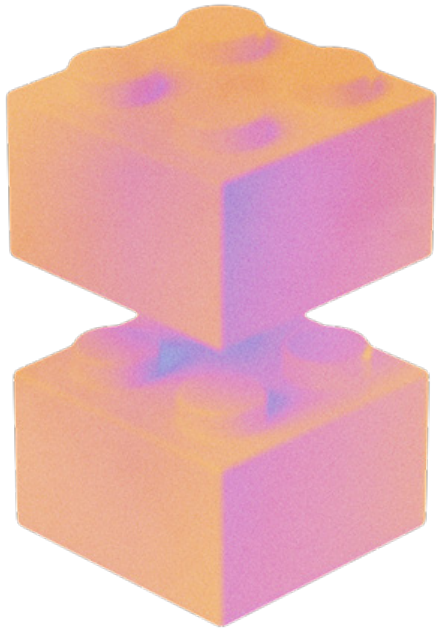
Traditional software executes rules. AI agents make decisions. We have moved from software that runs to software that acts. If a system can independently call APIs, write data, and trigger workflows, it is essentially a new digital employee, and the risk profile is now dependent on how they interpret their instructions and how they act when situations require judgement.

Traditional automation justified focusing on access because the behaviour that followed was predictable. Automation executed predefined instructions.

Agents operate with greater autonomy. They interpret tasks, select tools, and determine the sequence of actions needed to complete their work. Context shapes those decisions.

This creates a new governance requirement.

Governance must operate directly where these agent decisions happen, not rely on static policies reviewed after the work is complete.



3

Map every agent's full scope before it goes anywhere near production.

An agent's effective scope expands over time as tools are added and integrations grow. Each change improves utility. Over time the agent operates well beyond the system where it was originally defined. Document and review that scope continuously — not just at deployment.

Most enterprise agent systems do not remain inside a single platform.

A workflow that begins in a development environment may extend into SaaS applications, cloud infrastructure, APIs, and external services. Each system enforces its own controls, but the workflow itself spans several environments.

Agents are often described as platform-bound. In practice, they are anything but.

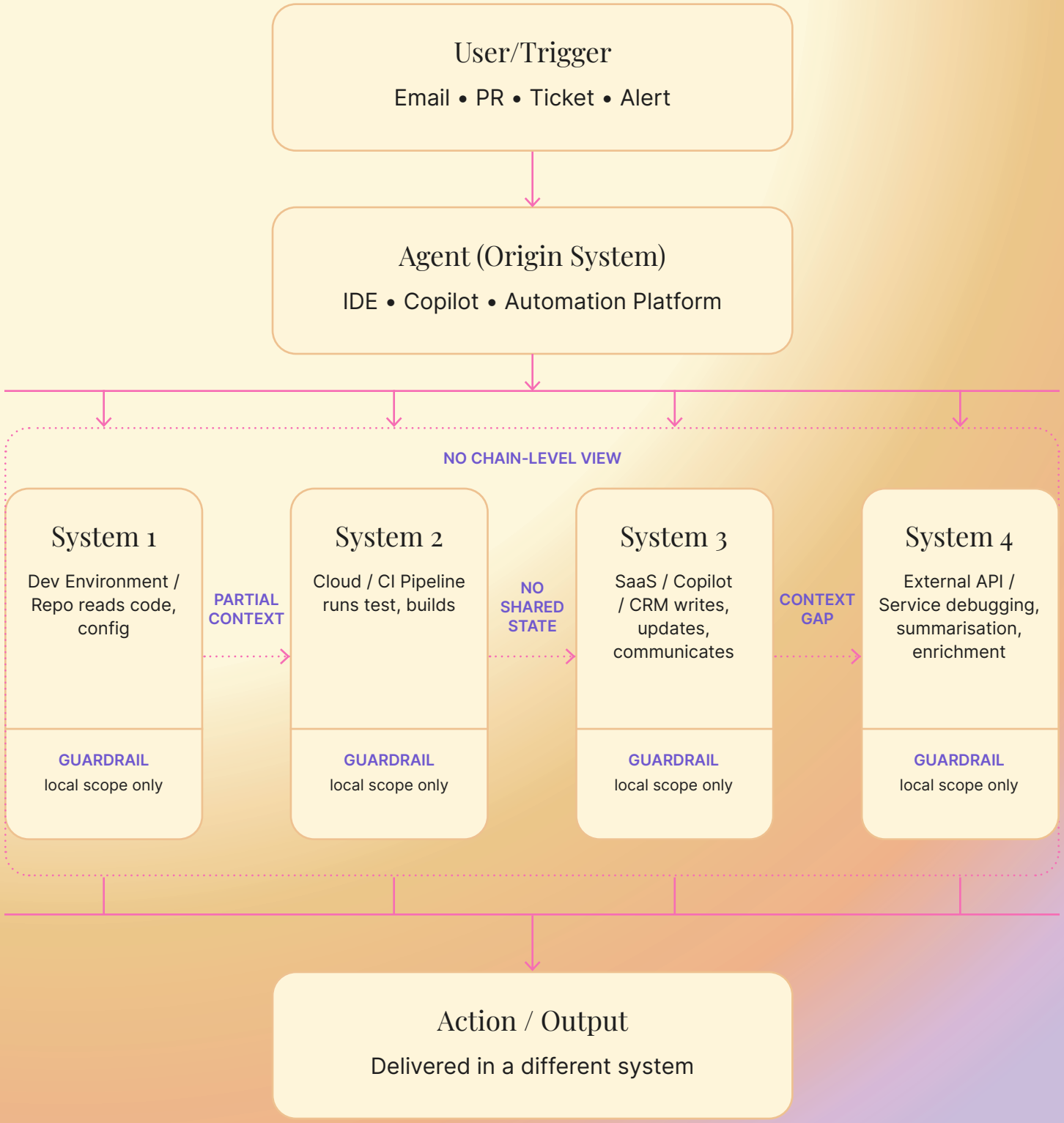
This transition rarely happens all at once. An agent starts with a narrow role and a small set of tools. As it proves useful, capabilities are added. New integrations are introduced. More context is made available to improve performance.

Each step is reasonable. Each change improves utility.

Over time, the agent's effective scope expands beyond the system where it was originally defined. Context retrieved in one environment influences behaviour in another. Actions taken in one system trigger consequences in the next.

From the perspective of any individual platform, the behaviour appears valid. Across the full workflow, the path becomes harder to reason about.

See the workflow on the next page.


User/Trigger
Email • PR • Ticket • Alert
Agent (Origin System)
IDE • Copilot • Automation Platform
NO CHAIN-LEVEL VIEW
System 1
Dev Environment / Repo reads code, config
GUARDRAIL
local scope only
PARTIAL CONTEXT
System 2
Cloud / CI Pipeline runs test, builds
GUARDRAIL
local scope only
NO SHARED STATE
System 3
SaaS / Copilot / CRM writes, updates, communicates
GUARDRAIL
local scope only
CONTEXT GAP
System 4
External API / Service debugging, summarisation, enrichment
GUARDRAIL
local scope only
Action / Output
Delivered in a different system

4

Design guardrails in — don't bolt them on after.

Guardrails are design decisions, not configuration settings. They define how an agent is allowed to operate before deployment. Response mechanisms detect problems after they emerge. Keep these concepts separate and invest in both.

The term *guardrails* appears frequently in discussions about AI governance. Its meaning often varies.

Guardrails refer to design decisions that define how an agent is allowed to operate before deployment.

Response mechanisms serve a different purpose. They detect behaviour and intervene after a problem has emerged.

Keeping these concepts separate improves clarity.

Guardrails are defined within systems. They are not designed to operate across them. Agent workflows, however, routinely cross those boundaries.

Different frameworks implement them in different ways. Some rely on prompt constraints. Others inspect tool calls. Some operate as middleware, while others exist primarily during development.

In a coding environment, an agent running through tools like Claude Code typically relies on prompt-level constraints, repository scoping, and local configuration. Enforcement depends heavily on how the agent interprets instructions in that context.

In a cloud or SaaS environment, such as a Copilot built in Microsoft Copilot Studio, guardrails are enforced through identity, connectors, and predefined action scopes tied to services like Microsoft Graph or internal APIs. Control is strongest within that ecosystem. Both approaches are effective within their respective boundaries.

The challenge appears when workflows span across them.

An agent may generate output in a development environment that is passed into a SaaS copilot. A cloud-based agent may call external APIs or trigger developer workflows. Each system enforces its own controls, but those controls do not extend across the full sequence of actions.

Even when guardrails are correctly implemented in each system, they remain blind to how decisions connect across them.

The result is fragmented governance. What is constrained in one environment may be unconstrained in the next, even though the workflow is continuous.

Guardrails as design

Guardrails define the operating scope of an agent before it is deployed. In many organisations they exist only within individual platforms, like your CI/CD tooling or your cloud environment, leaving cross-platform deployments without consistent standards.

Security teams often ask whether guardrails exist.

The more important question is whether they are consistent across every environment where agents run.

5

Assume your guardrails have gaps between systems.

Guardrails work within individual platforms. Agent workflows cross between them. What is constrained in one environment may be completely unconstrained in the next — even when every individual system is correctly configured.

The gap becomes clearer when looking at how agents operate in real environments.

In development workflows, a coding agent may retrieve code from a repository, run validation checks, trigger a build pipeline, and call an external debugging service.

Each action is authorised, yet the workflow extends beyond the environment where guardrails were originally defined. Context from the repository may appear in a debugging request sent to an external service. The request is legitimate. The data flow is not always intended.

In business workflows, an agent handling a customer request may retrieve account data, reference internal documentation, and call an external service to summarise or transform the response. Each step is permitted. If internal context is carried into that external call, sensitive information may leave the organisation without any single control being violated.

These outcomes do not come from a single incorrect action. They emerge from how agents combine context, tools, and decisions across systems.

Where Guardrails Stop Providing Full Coverage

Guardrails remain effective within the environments where they are defined.

They can validate prompts, constrain tool usage, and enforce clear boundaries inside a given system. That remains necessary. The limitation appears when workflows extend beyond those boundaries.

Enterprise agents routinely interact with external APIs, cloud services, and specialised tools. Once that happens, no single control point governs the entire sequence of actions.

The guardrails continue to function locally. They simply do not capture how the workflow unfolds across systems. From a security perspective, the system appears controlled while behaviour remains only partially understood.

The question becomes less about whether an action was allowed, and more about how a series of allowed actions produced an outcome.

6

Govern behavior across the full workflow — not just inside individual platforms.

The risk that matters emerges across systems — not within any single one. Agents act across development environments, SaaS platforms, cloud infrastructure and external services. Governance that stops at the platform boundary is governing architecture diagrams. Not operational behavior.

As agents move into real enterprise use, governance needs to reflect how they actually operate.

Agents act across development environments, SaaS platforms, cloud infrastructure, and external services. Each environment enforces its own controls, but the behaviour that matters emerges across them.

Security teams need to understand how decisions unfold step by step, which tools are invoked, how context moves between systems, and where authority is exercised.

Without that, organisations are governing architecture diagrams rather than operational behaviour.

The Next Phase of Agentic Governance

Early approaches to agent governance focused on model safety and prompt control. That reflected how these systems were first introduced: contained environments with clear boundaries.

That context no longer holds.

Agents operate across tools, systems, and services, carrying context forward and making decisions over time. The behaviour that matters emerges across those interactions, not within any single component. The boundary of control has moved. It no longer sits inside a single system, and it cannot be enforced from a single point.

Improving guardrails within individual platforms remains important. It strengthens local control and reduces obvious failure modes. It does not provide a complete view of how agents operate once workflows extend beyond those boundaries.

7

Establish a behavioral baseline for every agent.

You cannot detect drift without knowing what normal looks like. Two agents built on the same model can behave very differently depending on their prompts, context and toolsets. Build role-specific baselines — broad peer comparisons are not sufficient.

Anomaly detection always depends on a baseline.

Security teams already apply this principle in many areas. SIEM rules, network monitoring, and user behaviour analytics all rely on an understanding of normal activity. Agents require the same approach.

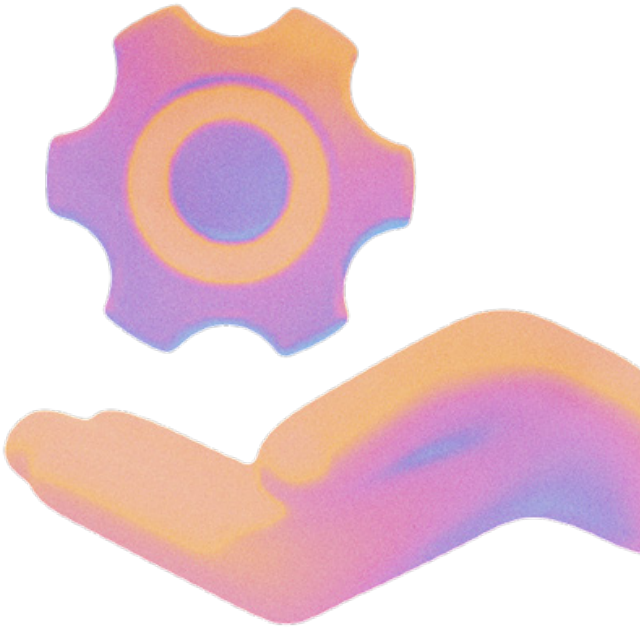
The difference lies in how that baseline is established.

Agent behaviour varies widely depending on the role, tools, and instructions assigned to the system. Two agents built on the same model can behave very differently if their prompts, context, or toolsets differ.

Meaningful baselines therefore need to reflect the specific role of each agent rather than broad peer comparisons.

Telemetry provides the foundation. Teams need visibility into the types of actions an agent performs, the tools it calls, and the frequency with which those actions occur. Over time this produces a picture of normal behaviour.

Once that baseline exists, deviations become easier to recognise.



8

Watch for drift caused by complexity — not just external threats.

The most common source of behavioral drift is not a bad actor manipulating the agent. It is an agent over-interpreting a complex multi-step task and extending its scope further than the organisation intended. Every individual decision can appear reasonable. The accumulated sequence can still produce an unintended outcome.

Many discussions of agentic risk focus on external manipulation. In practice, a more common source of drift is task complexity.

When an agent is asked to complete a multi-step task, it must select tools, interpret context, and decide how to reach the intended goal. Each decision may appear reasonable in isolation. The accumulated sequence can still extend beyond what the organisation intended.

This frequently happens when agents carry context between steps.

Information retrieved for one purpose can be reused during another stage of the workflow. Data that was appropriate for an internal analysis might later appear in a tool call or output that reaches a wider audience.

Every step can be legitimate. The overall outcome can still be problematic.

Drift often begins with complexity.

The more decisions an agent must make to complete a task, the greater the chance that it interprets its scope too broadly.

Understanding normal agent behaviour allows teams to recognise when activity moves outside expected patterns. In many environments, drift comes from agents over-interpreting complex tasks and extending data boundaries beyond what was intended.

Behavioural baselines cannot be established once and then forgotten.

Agents evolve continuously. Tools are added. prompts change. Tasks become more complex. These adjustments often happen without formal change management.

By the time an incident occurs, the behaviour that produced it may have developed gradually over weeks.

Continuous telemetry and monitoring therefore become a core governance function. Drift detection operates as an early signal that behaviour is moving outside expected patterns.

9

Treat your system prompts as governance infrastructure — not local configuration.

System prompts are the agent's employee handbook. They describe the agent's role, establish expectations and define the standards it must follow. Most organisations treat them as a developer's local concern. Security and governance teams rarely see them at all. That gap needs to close.

One of the most effective guardrails available today is the system prompt.

A well-written prompt functions as the agent's handbook. It describes the agent's role, explains how tasks should be performed, and defines boundaries around acceptable behaviour.

Agents respond best to explicit instructions. Vague guidance provides little direction during ambiguous tasks. Statements such as "follow company policy" leave interpretation entirely to the agent.

Clear instructions produce more predictable behaviour.

Many guardrail failures originate from unclear prompts or from the absence of a prompt altogether.

An agent without a defined prompt lacks a reference point for interpreting requests. In that situation the agent infers its scope from the task it receives.

Explicit prompts provide the structure required for consistent decision-making.

System prompts define the operating contract between an organisation and its agents.

They describe the agent's role, establish expectations, and provide the standards the agent should follow when interpreting instructions. In effect, they function as the agent's employee handbook.

Without this guidance, an agent must infer how it should behave.

Most organisations treat prompts as local configuration rather than shared infrastructure. A developer writes the prompt during deployment. Another team may modify it later. Multiple versions appear across environments without central visibility.

Security and governance teams rarely see these prompts at all.

This creates a gap between the intended behaviour of an agent and the instructions it actually receives.

10

Make sure every agent has a prompt that actually describes its job.

Vague instructions like “use good judgement” or “follow company policy” leave interpretation entirely to the agent. Every agent needs a specific, narrow description of its role, the actions it may perform and the actions it must never take. General instructions encourage general interpretation.

The most common problem is vagueness.

Instructions such as “use good judgement” or “avoid violating company policy” offer little direction when an agent encounters an ambiguous request. The agent must interpret those instructions on its own.

Clear prompts describe specific tasks, boundaries, and expectations.

They define the role of the agent, the actions it may perform, and the actions it should avoid. They also reflect the organisational standards that apply to that work.

Agents operate most predictably when their responsibilities are narrowly defined and supported by focused context.

Prompts work best when they describe a specific job with clear boundaries.

General instructions encourage general interpretation.

What enforcement actually means

Prompt enforcement extends beyond verifying that prompts exist.

The goal is to understand whether prompts produce the behaviour they were intended to guide. This requires observing how agents actually act and comparing that behaviour to the instructions they were given.

When agents consistently perform actions outside their described role, the gap becomes a governance signal.

Over time, organisations can refine prompts so that they more accurately reflect the tasks agents perform in practice.

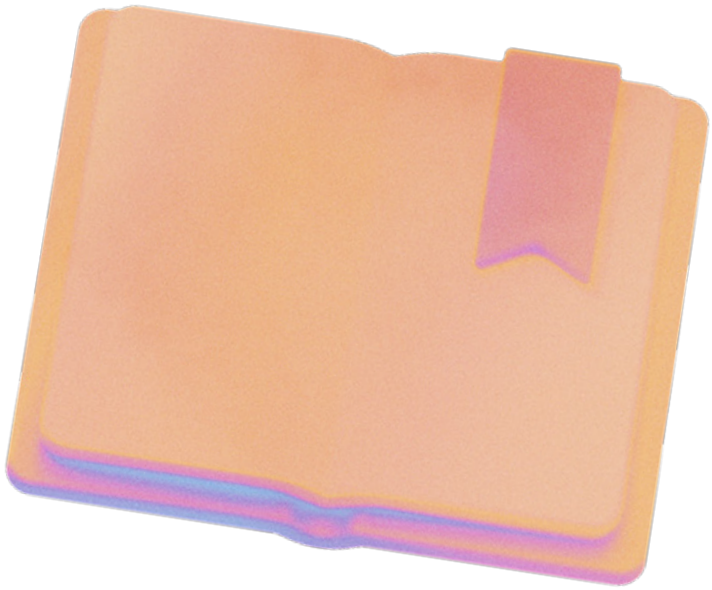
Most enterprises begin with a small set of shared standards.

These often take the form of organisational “always” and “never” rules that apply to all agents. Examples may include restrictions around data handling, communication with external systems, or escalation requirements.

Every agent should then have a prompt that describes its specific role within those broader standards.

As agent usage grows, prompts become more detailed and tailored to the responsibilities of individual systems.

The objective is consistency in principles and specificity in execution.



Prompt enforcement

System prompts function as the agent’s employee handbook. They describe the agent’s role and the standards it must follow. Organisations need visibility into how prompts guide behaviour and whether agents follow those instructions in practice.

What Good Looks Like: a Maturity Model for Agent Governance

Agent governance matures over time.

Most organisations begin with a small number of experimental agents and informal oversight. As deployments expand, governance becomes more structured.

Three areas define that progression: guardrail design, behavioural baselines, and prompt governance. Together they provide a clearer picture of how agents operate and how their behaviour aligns with organisational expectations.

A Governance Maturity Framework

Level	Guardrails	Baselines & Drift	Prompt Posture
Foundational	Prompts absent or vague. Guardrails defined per platform only.	No formal behavioural baseline. Telemetry limited.	Prompts written during deployment with little review.
Developing	Prompt templates and basic organisational standards introduced.	Early telemetry analysis begins identifying common drift patterns.	Standard prompts used for common agent roles.
Operational	Guardrail design separated from response mechanisms. Signals inform risk classification.	Role-specific baselines established. Drift detection begins identifying tool and data anomalies.	Behaviour compared against prompt intent to identify divergence.
Advanced	Governance standards applied consistently across platforms and agent frameworks.	Continuous drift detection supported by behavioural telemetry.	Prompts tailored to each agent while maintaining organisational standards.

Where to start

A practical starting point requires two steps.

Together these steps create a shared reference point for both the agent and the organisation.

1. **First, ensure every agent has a prompt that clearly describes its role and expected behaviour. Generic disclaimers do not provide sufficient guidance.**

2. **Second, establish organisation-wide behavioural standards that apply to every agent regardless of platform.**

What separates organisations that are ahead

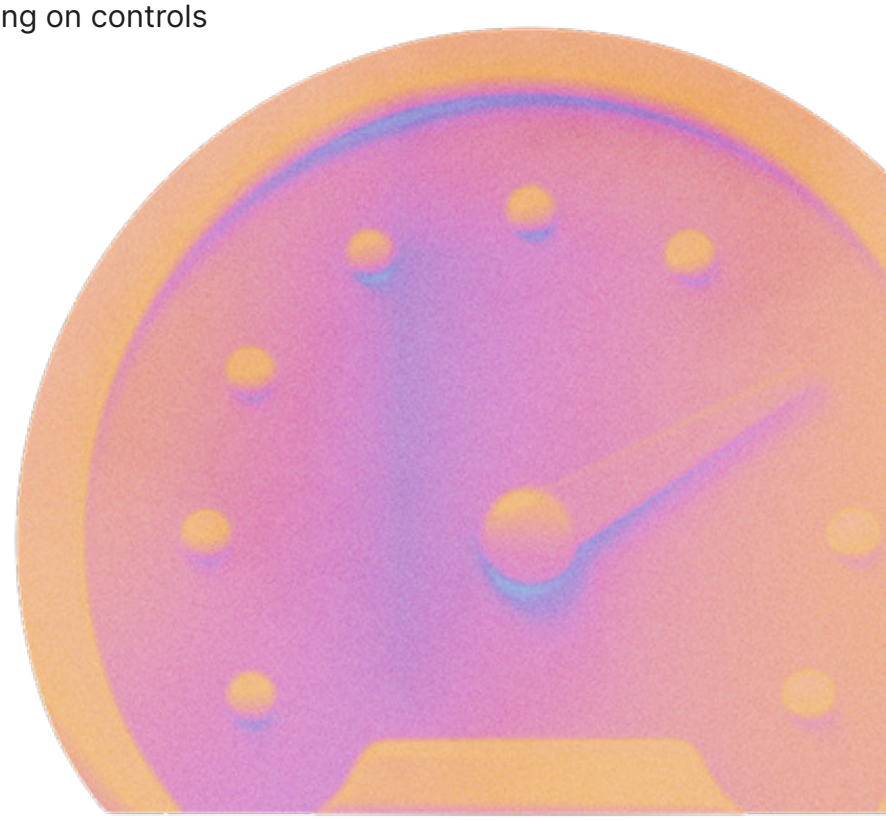
The organisations that progress fastest recognise that agent governance cannot be confined to a single platform.

Agents appear across developer environments, low-code automation tools, SaaS platforms, and cloud infrastructure. Each environment provides its own controls and visibility.

Effective governance therefore focuses on behaviour across those environments rather than relying on controls within any single one.

This perspective shapes how teams design policies, collect telemetry, and review agent activity.

Agents will continue to expand across enterprise systems. Organisations that understand how those agents behave will be better positioned to adopt them with confidence.



Conclusion

The ten practices in this guide are not theoretical. They reflect what separates organisations that are scaling agents with confidence from those that are discovering risks after the fact.

None of them require starting over. They do not replace the identity and access controls already in place. They extend governance into the territory those controls cannot see — behavior, sequence, context and drift.

The starting point is simpler than it looks. After visibility as absolute ground zero, make sure every agent has a prompt that describes its job. Establish behavioural standards that apply across every environment where agents run. Build from there.

Agents will continue to expand across enterprise systems. The organisations that understand how those agents behave will be better positioned to adopt them — and better positioned to defend them when it matters.