

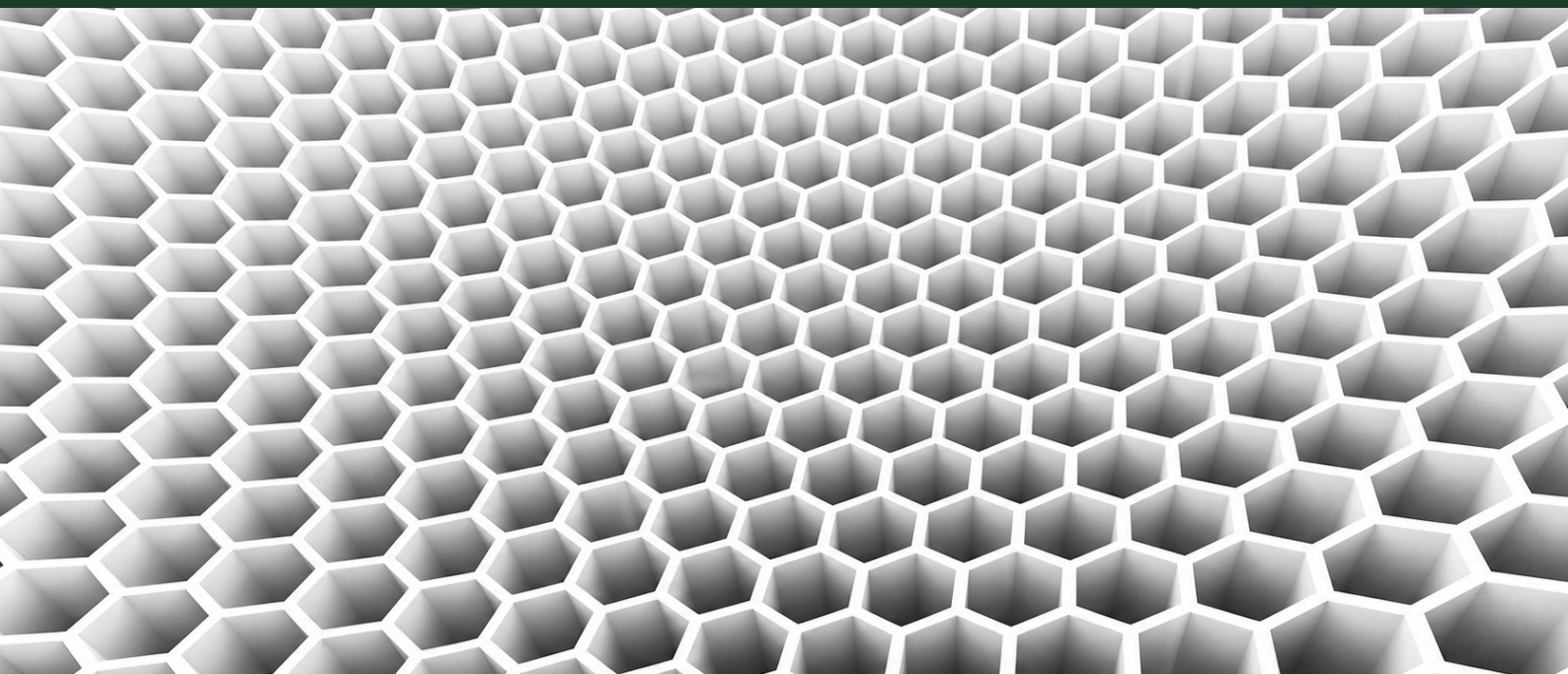
BEYOND THE VENDOR DECK

Agentic AI for the Delivery Leader — A Primer Without the Hype

A plain-language guide to what agents are, what they change, and what to ask before they reach your programme

Giovanni Leonardi

July 2025



Agentic AI for the Delivery Leader — A Primer Without the Hype

Programme · No. 236
July 2025

"The agent does not know the organisational consequences of its actions. It knows the API."

1. The Word You Keep Hearing

The technology proposal lands on your desk on a Tuesday afternoon. Somewhere between the architecture diagram and the cost breakdown, the word appears: agentic. The vendor deck uses it eleven times. Your CTO mentioned it last week. The consultancy's slide pack promises it will "transform delivery." And you — the person who will be accountable when this reaches the programme board — cannot yet describe what it actually means.

You are not alone, and you are not behind. The term has outrun its definition. What follows is an attempt to close that gap: a plain-language account of what an AI agent is, what it changes operationally, what goes wrong in practice, and what you need to have in place before one runs inside your programme. Nothing here requires a computer science degree. Everything here is written for the person who needs to make sound governance decisions about a technology that is genuinely new, not merely rebranded.

2. What an Agent Actually Is

Strip away the marketing and an AI agent is a large language model — the same technology behind the chatbots your organisation is already experimenting with — given three additional capabilities: goals, tools, and memory. Where a chatbot waits for a prompt and returns a single answer, an agent receives an objective and works towards it in a loop, deciding at each step which tool to use, executing it, reading the result, and deciding what to do next.

The loop is the defining feature. A chatbot is a single exchange. An agent is a sequence of exchanges that the model itself orchestrates, sometimes running dozens of steps before returning a result. It might query a database, draft a summary, check it against a policy document, revise, and send an email — all from a single instruction.

This is a genuine architectural shift, not a rebranding exercise. But it is also a shift that is easier to describe than to govern, and the gap between description and governance is where the risk sits.

2.1 The three components in plain language

- Goals — the instruction that tells the agent what to achieve. This can be as narrow as "reconcile these two spreadsheets" or as broad as "monitor this programme's risk register and flag emerging issues." The breadth of the goal is the single largest determinant of how much autonomy the agent has, and therefore how much can go wrong.
- Tools — the systems the agent can interact with. An agent without tools is just a chatbot thinking out loud. Tools are what give it reach: the ability to read files, query databases, call APIs, send messages, create documents, or trigger workflows. Every tool is a surface area for error.
- Memory — the context the agent carries between steps, and sometimes between sessions. This includes what it has already done, what it has read, and what instructions it is operating under. Memory is finite and imperfect — the agent can lose track of earlier steps in a long sequence, and this is a failure mode, not a bug.

3. What Changes Versus Classic Automation

Programme directors are used to governing automation. RPA bots, workflow engines, scheduled scripts — these follow deterministic rules. They do the same thing every time, and when they fail, they fail in predictable ways. Agents are different in four respects that matter for governance.

3.1 Probabilistic output

Give an agent the same input twice and it will not necessarily produce the same output. This is inherent to the underlying language model, not a defect. It means that testing an agent is closer to testing a human process than testing software: you can verify the quality of outputs across a sample, but you cannot guarantee identical results on every run.

3.2 Context limits

Every agent operates within a finite context window — the amount of information it can hold and reason about at any one time. When the task exceeds that window, the agent does not stop and ask for help. It works with what it has, which means it may silently ignore information that it should have considered. For programme-scale work involving large documents, complex histories, or multiple data sources, this is a material constraint that no amount of vendor optimism changes.

3.3 Compounding errors

In a deterministic system, an error in step three is an error in step three. In an agentic loop, an error in step three becomes the input for step four. The agent treats its own output as fact and reasons forward from it. A small hallucination — a figure misread, a date transposed, a policy clause paraphrased inaccurately — can propagate through a chain of subsequent steps and arrive as a confident, well-structured, and entirely wrong

conclusion.

3.4 Cost per action

Classic automation costs money to build and almost nothing to run. Agents cost relatively little to build but charge per action — every step in the loop consumes tokens, every tool call may incur API costs, and a poorly scoped task can run up significant charges before anyone notices. The cost model is closer to a utility bill than a capital investment, and it requires monitoring that most programme offices have not yet set up.

4. The Failure Modes to Plan For

Vendor demonstrations show agents succeeding. Governance requires you to plan for agents failing. Three failure patterns recur across early deployments, and none of them is exotic — they are the predictable consequences of the architecture described above.

4.1 Silent drift

The agent is working. It is producing output. It has not thrown an error. But somewhere in its reasoning chain it has drifted from the original intent, and it is now optimising for something subtly different from what was asked. This is the most dangerous failure mode because it looks like success until someone examines the output closely. The status report that is well-written but draws on the wrong data source. The risk assessment that sounds authoritative but has quietly dropped a category. The summary that is fluent and persuasive and factually adrift.

4.2 Tool misuse

The agent has access to tools and the autonomy to decide when to use them. In the normal case, it uses the right tool for the right purpose. In the failure case, it uses a tool in a way that is technically valid but contextually wrong — sending an email that should have been a draft, updating a production record when it should have updated a test environment, querying a system with parameters that return misleading results. The agent does not know the organisational consequences of its actions. It knows the API.

4.3 Runaway loops

The agent encounters a step it cannot complete and, rather than stopping, retries with variations. Each retry costs tokens and time. In a well-designed system there are circuit breakers — hard limits on iterations, cost ceilings, timeout thresholds. In a hastily deployed system, the agent can run hundreds of iterations, consuming budget and sometimes generating cascading effects in connected systems, before anyone intervenes. We have seen this pattern in early pilots and it is almost always a design failure, not a model failure: no one defined the boundary.

5. Questions to Ask Any Vendor or Team

When an agent is proposed for your programme — whether built internally or offered by a vendor — the following questions separate a considered deployment from a hopeful one. They are not technical questions. They are governance questions, and the inability to answer them clearly is itself an answer.

5.1 On autonomy scope

1. What is the agent authorised to do, and what is it authorised to decide? These are different things. An agent can be authorised to draft a report without being authorised to decide which data sources to include.
2. What is explicitly out of scope? The boundary should be defined by what the agent cannot do, not only by what it can.
3. Who approved this scope, and when is it next reviewed?

5.2 On reversibility

1. Can every action the agent takes be undone? If not, which actions are irreversible, and what approvals gate them?
2. Is there a human-in-the-loop checkpoint before any irreversible action?
3. What is the rollback procedure if the agent produces incorrect output that has already been acted upon?

5.3 On the audit trail

1. Is every step in the agent's reasoning chain logged — not just the final output, but each intermediate decision, tool call, and result?
2. Can a reviewer reconstruct why the agent did what it did, after the fact?
3. How long are logs retained, and who has access?

5.4 On the kill mechanism

1. Can the agent be stopped mid-task? By whom? How quickly?
2. What happens to in-flight work when the agent is stopped — is it saved, discarded, or left in an indeterminate state?
3. Is there an automatic shutdown trigger if cost, time, or error count exceeds a threshold?

6. Governance Minimums Before Production

No governance framework for agentic AI is settled yet — we are all working this out as the technology moves. But early deployments across sectors have produced a set of minimums that consistently separate the programmes that manage agents well from those that manage them badly. None of these requires new technology. All of them require the same discipline that good programme governance has always required: clarity about who is accountable, what is permitted, and how you know it is working.

The governance minimum for an AI agent is no different in kind from the governance minimum for any autonomous process: define the boundary, log the actions, review the outputs, and keep the kill switch within reach.

Area	Minimum requirement	Who owns it
Scope definition	Written statement of what the agent may do, may decide, and may not do — reviewed before deployment and at fixed intervals	Product or delivery lead
Human-in-the-loop	Defined checkpoints where a human reviews and approves before the agent proceeds — mandatory for any irreversible action	Process owner
Output review	Scheduled sampling of agent outputs against ground truth — checking not just for errors but for drift	Quality or assurance lead
Cost monitoring	Token and API cost tracking with alerting thresholds, reviewed weekly	Finance or technology lead
Audit logging	Full chain-of-reasoning logs retained for an agreed period, accessible to governance and audit functions	Technology lead
Kill switch	Documented procedure to halt the agent, known to at least two named individuals, tested before go-live	Operations or delivery lead
Incident response	Defined escalation path for agent failures, integrated into existing incident management processes	Delivery or risk lead
Review cadence	Standing review of agent performance, scope, cost, and incidents — monthly at minimum	Programme board

7. What You Can Safely Set Aside

A primer's value lies not only in what it covers but in what it tells you to deprioritise. Three topics dominate the conference circuit and the vendor decks, and none of them needs to be on your agenda this quarter.

- Multi-agent orchestration — systems of agents coordinating with each other autonomously. The concept is real and the research is active, but production deployments today overwhelmingly involve single agents with defined tool sets. Design governance for one agent at a time. That is hard enough and it is the right place to start.

- Autonomous decision-making at scale — the proposition that agents will soon replace human judgement across entire business processes. We are not close to this, and the programmes that attempt it before the governance foundations are in place will generate the cautionary case studies. Begin with agents that assist and draft, not agents that decide and execute.
- The existential debate — whether agentic AI represents a fundamental risk to employment or society. These are important conversations. They are not useful conversations for the programme board meeting where you need to decide whether to approve a deployment. The governance questions are practical, and the practical questions are hard enough to deserve your full attention.

8. A Note on Pace

The temptation is to wait — to let the technology mature, the standards settle, the case law develop. This is understandable but it misreads the situation. The risk is not that you adopt too early; it is that you adopt without understanding, because the technology will arrive in your programme whether you planned for it or not. It will arrive in a vendor's platform update, in a team's productivity tool, in a consultant's methodology, in a junior analyst's side project.

The delivery leader who understands what an agent is, what it changes, and what to ask is not the one who adopts fastest. They are the one who governs well when adoption happens — and who can explain to the programme board, on Monday morning, exactly what they are governing and why.

That starts with the questions in this primer. They are not comprehensive and they will need updating as the technology and the governance landscape evolve. But they are enough to be accountable on Monday.