Andy Hickman
ADVISORY

A PRACTICAL PLAYBOOK

Adopting Agentic AI

> How to choose, build, govern and scale AI agents that genuinely improve business workflows

INSIDE

— Key takeaways
The nine points to take away

01 Introduction
What an agent is, and the question that decides every use case

02 Three Foundational Ideas
When to use an agent at all, how to verify its work and how much autonomy to give it

03 The Lifecycle Playbook
The core: seven stages to follow for each use case you automate

04 Governance
Security, auditability, reliability and ownership

05 Cost
Where the cost comes from and how to control it

06 People and Change
Involving users, communication and training

07 Adoption Roadmap
A phased path from first pilot to scale

Andy Hickman · 15 October 2026

andy-hickman.co.uk

SUMMARY

Key takeaways

01 **Verifiability is a must.** The best agentic use cases are ones where you can reliably check the agent's work. If you can't check it, you can't trust it.

02 **You may not need an agent at all.** Traditional automation or a fixed AI-assisted workflow can be cheaper and more predictable. Use the simplest approach that works.

03 **Start boring.** Your first agent should be high-volume, internal, low-risk and easy to verify, not your flagship process.

04 **Identify the test set (evals) early.** A few real examples with known-good answers are highly valuable and clarify thinking.

05 **Autonomy is earned.** Start with humans approving outputs and increase autonomy only as evidence accumulates.

06 **A working prototype is not a finished product.** The prototype is roughly 20% of the effort; making it secure, reliable, widely available and ready for everyday use is the other 80%. Make the move to production a deliberate decision.

07 **Govern it like a new, inexperienced member of staff with system access.** That means its own identity, minimum permissions, a full audit trail and a named owner.

08 **Introducing agents is a change programme, not just a technology task.** Close end-user involvement, clear communication and training decide whether an agent is adopted or quietly ignored.

09 **Don't forget the cost of ownership.** Like any other IT system, an agent has ongoing running and maintenance costs, and a continuing need for technical skills.

SECTION 01

Introduction

This playbook distils lessons from my work helping organisations adopt agentic AI: what separates the pilots that reach production and deliver value from the many that stall. It is written for leaders and managers shaping AI strategy or looking to improve how their teams work, and for the technical teams who will build and run the agents. The main text is deliberately non-technical; boxes marked **For your tech team** carry the implementation detail.

What is agentic AI? An AI agent pursues a goal with limited human intervention by deciding for itself what actions to take and judging when the work is done. It gathers information, decides what to do, uses tools or systems (for example, searching the web or a database, or updating a support ticket), checks the result and repeats as needed until the work is done. An agent may run an entire process, or a single stage within a larger, fixed one. That makes it different from a single AI call, which answers one question, and from a traditional software application workflow, which follows a predetermined sequence of steps. The difference matters because it determines both where agents add value and where they add risk.

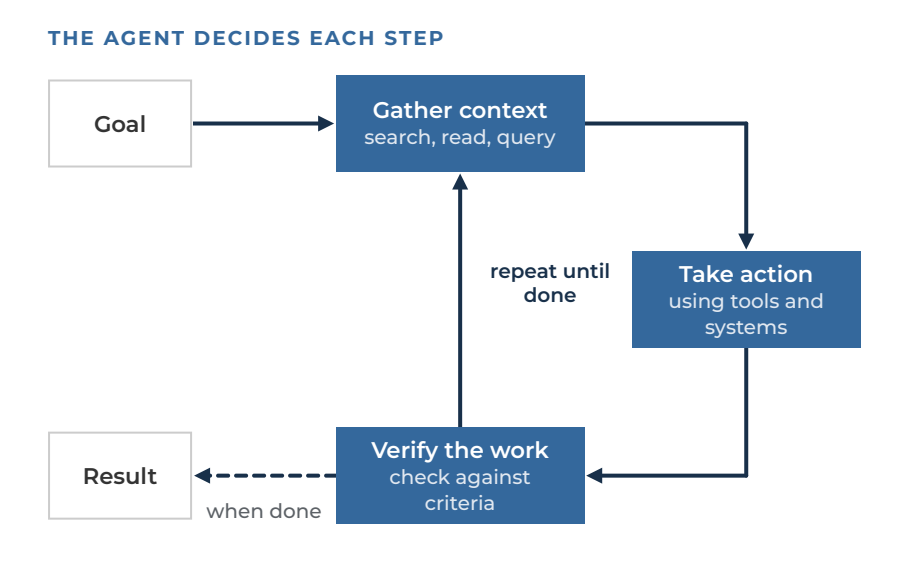

Figure 1. The agent loop: gather context, take action, verify, repeat until done.

The defining question for any potential use case is simple: *can you check the agent's work reliably enough to trust it?* Section 2.2 explains how.

SECTION 02

Three Foundational Ideas

2.1 Do you need an agent at all?

Agents are powerful but they are not always the right tool. They cost more to run than conventional software, they are slower, and they are non-deterministic: the same input will not always produce exactly the same output. Before choosing an agent, work through the options from simplest to most capable and stop at the first one that does the job.

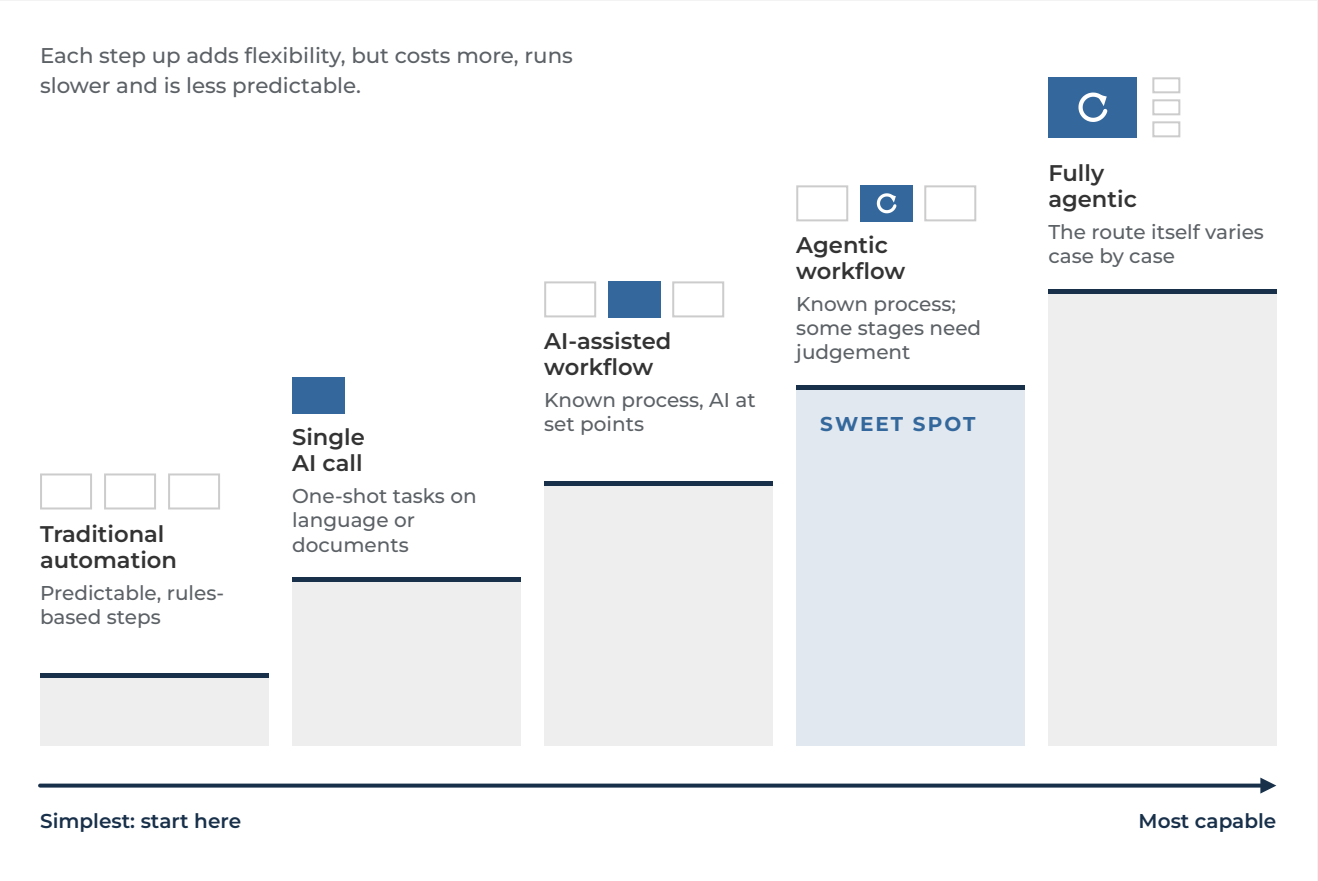

Figure 2. Choose the simplest approach that does the job. White boxes are conventional code, blue boxes are AI, and the loop marks an agent choosing its own actions.

For most business processes, the **agentic workflow** is the sweet spot: the organisation keeps control of the overall process, conventional code handles the predictable steps, and autonomy is given only where judgement is needed. For example, a fixed market-research workflow might include one agentic stage that searches the web for a company's revenue and market, returning the figures and their sources in a structured format. The agent decides what to search for and when

it has a credible answer, and the sources make each answer easy to check.

An agent is the wrong choice when:

- **The logic is deterministic.** If the task can be written as ordinary rules, write the code or use existing automation tools such as robotic process automation (RPA).
- **There is no way to check the output and the stakes are high.** Add a verification step first, or don't use an agent.
- **The rules are undocumented and keep changing.** If the logic lives only in people's heads, stabilise and document the process first.

2.2 Verifiability is a must

There are three ways to check an agent's work:

- **Automated checks**, such as totals reconciling, data passing validation or tests passing. These are the strongest and cheapest because they run on every output, and they are what allows higher levels of autonomy.
- **Human review** against clear criteria, such as approving a draft. This is flexible, but it costs reviewer time.
- **AI-as-judge**, where a second AI model scores the output against defined criteria. This sits in between, but it is itself imperfect and should be calibrated against human judgement.

If success can be verified in at least one of these ways, you have a strong candidate. If success is subjective, open-ended or impossible to check, an agent will be hard to trust and harder to operate.

Verification is the thread that runs through this playbook. It determines whether a use case is suitable (Stage 1), what the prototype is measured against (Stages 2 and 3), how much autonomy is safe (Section 2.3) and how quality is maintained over time (Stage 7). It is also what lets an agent correct its own mistakes as it works. Wherever you see a decision in this playbook, the first question is usually: *how will we know the agent got it right?*

2.3 Autonomy is a dial

Autonomy is not a yes/no question; the question is how much of it a workflow can safely support. Classify each workflow on a scale and design the human involvement to match.

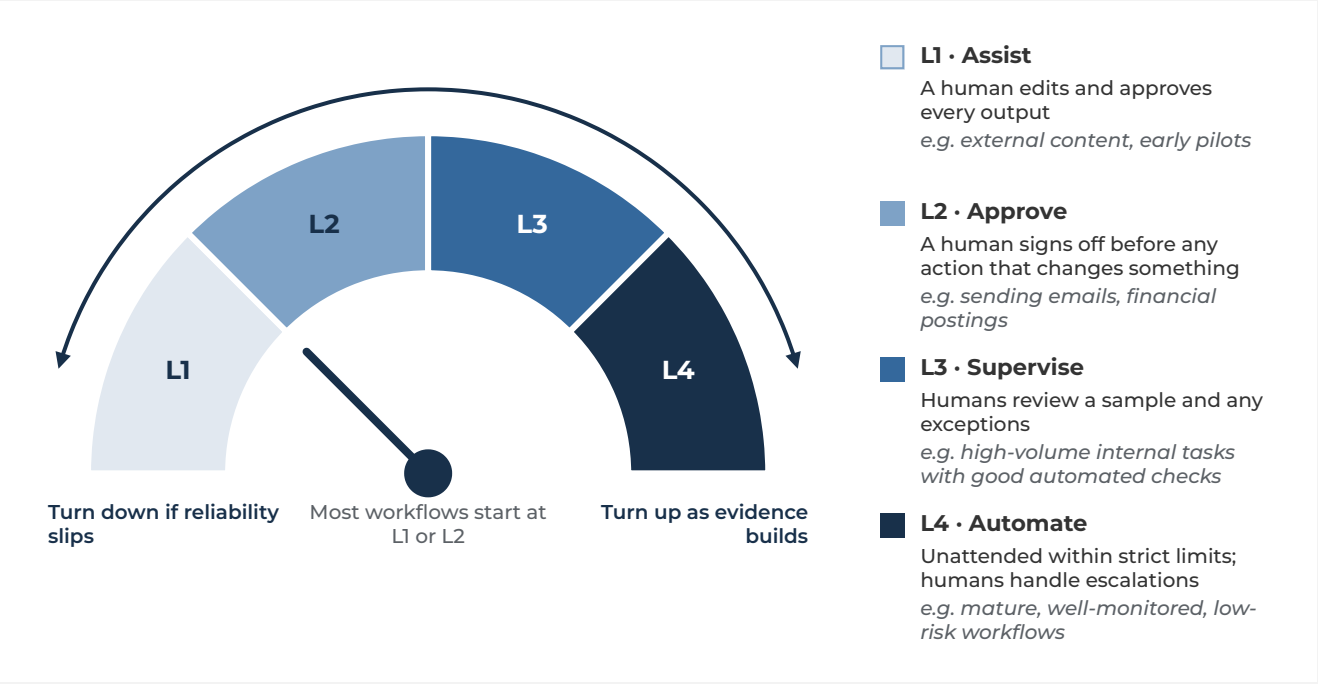

Figure 3. Autonomy is a dial: start most workflows at L1 or L2, and turn it up only as evidence builds.

SECTION 03

The Lifecycle Playbook

This section is the operational core. Follow it for each use case you automate. The stages are iterative: expect to loop back, especially between design, prototype, build for production and test.

Depending on your context, the stages don't need to be formal: a light-touch or even verbal approach can be fine, but take a moment to think through the considerations of each stage as you go.

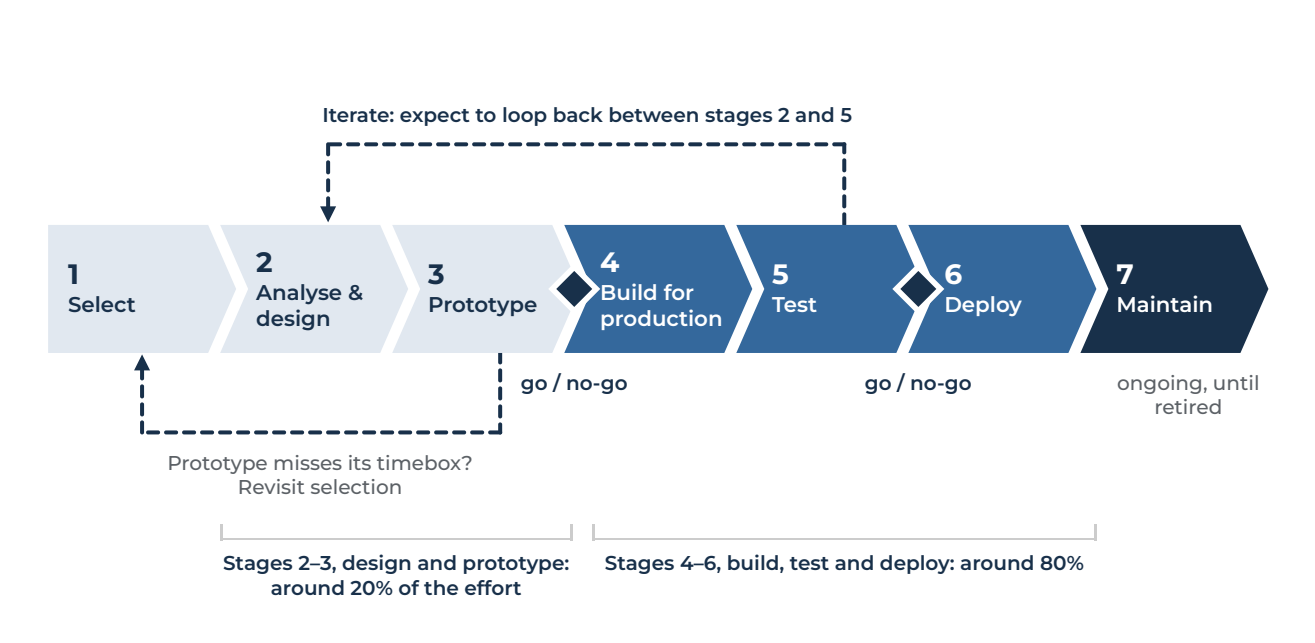

Figure 4. The seven stages. Expect to loop back between them, and make the move to production a deliberate decision.

3.1 Stage 1 - Select

1 2 3 4 5 6 7

Lifecycle · stage 1 of 7

The most expensive mistake in agentic AI adoption is starting with the wrong use case. Most failed pilots fail not because the technology couldn't do the job, but because the chosen task was impossible to verify, low in value, or too risky (closely tied to irreversible real-world actions that could be costly). Selection is a joint business and technical decision.

Assess candidates against these criteria. The first three are gating: if a workflow fails any of them, set it aside however attractive it looks.

Criterion	What good looks like
Verifiable (gating)	Output can be checked against clear criteria using the methods described above
Bounded (gating)	A finite set of tools, data sources and decisions, and a clear point at which the task is done
Recoverable (gating)	Mistakes can be undone; significant or risky actions can be held for human approval (the "blast radius" of an error is contained)
Valuable	High volume, repetitive, or high labour cost per instance
Data available	The information the agent needs is accessible through systems or documents, and those sources are stable and high quality
Tolerant of delay	Doesn't need an instant response, or can run in the background
Stable rules	The business logic can be written down and doesn't change daily

Run a short scoring workshop with business and technical stakeholders, and include people who do the work today:

1. Confirm the three gating criteria. If any fails, park the workflow.
2. Score value (1–5): how much time or cost does one instance consume, and how often does it occur?
3. Score feasibility (1–5): data/system access, tolerance of delays and rule stability.
4. Agree the autonomy level the task can safely support (Section 2.3).
5. Plot value against feasibility and prioritise the high-value, high-feasibility quadrant.

Start boring. Your first agentic project should be boring, high-volume, internal and easy to verify, such as drafting first-pass responses to a structured ticket queue, reconciling two data sources, or triaging and labelling incoming documents. Save customer-facing, high-stakes or creative work until you have operational confidence.

Then define what success looks like for the business. Agree the targets the agent must hit to be worth deploying, such as accuracy, cost per task, turnaround time and user satisfaction. This is about *what* good enough means; Stage 2 works out *how* it will be measured. It isn't just a technical exercise: it gets all stakeholders aligned on what matters.

Secure stakeholder buy-in, with a named business owner and a named technical owner.

Pitfalls. Avoid the *flagship trap*: resist starting with your most visible, highest-stakes process. Also check data access early. A workflow isn't feasible if the agent can't reach the information it needs.

READY TO MOVE ON WHEN Stage 1 of 7

- [ ] A set of prioritised use cases with documented success criteria
- [ ] An agreed autonomy level
- [ ] Confirmed data access
- [ ] Named business and technical owners

3.2 Stage 2 - Analyse & Design

1 **2** 3 4 5 6 7

Lifecycle · stage 2 of 7

This stage is not a waterfall step to be fully specified and signed off before anything is built. It contains two kinds of work: a short list of **gates** to settle before prototyping, and a brief **design** that evolves alongside the prototype rather than being perfected on paper.

Gates: settle these before building

- **Decide how the agent's work will be judged, and start to build the test set (evals).** Stage 1 established that the work *can* be verified; now choose the specific mechanism (an automated check, a named human reviewer or an AI-as-judge rubric). Then assemble real examples of the task with known-good outcomes: the evaluation, or "eval", set. These are among the most valuable thing you will create: start with a few examples and build up as you go along. They turn the Stage 1 success targets into concrete tests, let you compare changes objectively and become the basis of ongoing quality testing, and the same checking mechanism later runs on live work in production. Forgetting the eval set is the most common mistake I see: teams end up tuning and judging the agent in an ad-hoc, unrepeatable way. If you can't define how outputs will be judged, don't prototype yet; go back to Stage 1.
- **Set the data and security boundary.** Identify what data the agent will read and write, how sensitive it is, and any residency or redaction requirements. Decide what access the prototype will have and the environment it will run in.

Scale the gates to the risk. For a low-risk internal pilot on non-sensitive data, the security gate is a quick confirmation, not a project; for external-facing or regulated workflows, it is substantial.

Design: sketch now, revise during prototyping

- **Map the current process with the people who do it.** Document the steps, decisions, inputs, outputs and exceptions. Separate the predictable steps (candidates for conventional automation) from those needing judgement. Don't faithfully reproduce legacy workarounds and quirks; this is a chance to simplify.
- **Sketch how the agentic parts will work.** Every well-designed agent follows the same three-part pattern: *gather context, take*

FROM MY EXPERIENCE

One of my clients was seeking to automate an ERP upload of various forecast data that involved multiple manual steps. By whiteboarding the workflow, we realised that conventional automation was likely to be the right answer, and further building confirmed that.

FOR YOUR TECH TEAM

For each tool, specify a precise name, description, input/output schema and side-effect classification (read-only vs side-effecting). Prefer letting the agent retrieve context itself (search, file access, API calls) over stuffing everything into the prompt. Fewer, well-described tools beat many overlapping ones. Expect the prototype to add, remove and merge tools. Put the eval set under version control early on.

action, verify the work, then repeat until done. Design each part explicitly.

- **List the tools and data sources** it will need, distinguishing those that only read information from those that change something.
- **Draft the guardrails**: limits on time and cost per task, approval points and escalation routes. The specific limits will be tuned from prototype data.

READY TO MOVE ON WHEN Stage 2 of 7

- [ ] An agreed way of judging outputs, with a named owner
- [ ] An eval set of real examples with known-good outcomes
- [ ] The data and security boundary set in proportion to risk
- [ ] An initial design, understood as a starting hypothesis rather than a frozen specification

3.3 Stage 3 - Prototype

1 2 **3** 4 5 6 7

Lifecycle · stage 3 of 7

A prototype answers one question as cheaply as possible: *can an agent do this task reliably enough to be worth building for production*? Optimise for speed of learning, not code quality or scale. Most of what you build will be thrown away, and that is the point. Expect the prototype and the design to evolve together.

- **Build the thinnest possible version** using an established agent framework rather than building the machinery from scratch.
- **Use a capable model**, so you're testing the task rather than fighting model limitations. Optimise for a cheaper model later.
- **Run any action that changes something in a "dry run" or staged mode**, so nothing real is affected.
- **Log everything from early on.** You cannot improve, cost or audit an agent you cannot observe.
- **Measure against the eval set** and iterate on the instructions, tools and context.
- **Agree the exit criteria in advance**: the success rate, cost and turnaround time that would justify going further, and what would kill the idea. Writing these down first keeps the go/no-go decision evidence-based rather than emotional.

Timebox the prototype. Cap the initial prototype at one to three weeks. If you can't show promising results against the eval set in that window, the task is probably too hard, too open-ended or poorly specified. Revisit Stage 1 rather than pushing on.

FOR YOUR TECH TEAM

Use an agent framework or SDK for the loop, tool execution and context management. Call models via a framework or API (e.g. OpenRouter) that makes swapping models trivial and avoids vendor lock-in. Connect systems through an open standard such as the Model Context Protocol (MCP). Log reasoning, every tool call with its inputs and outputs, tokens used and the verification result.

READY TO MOVE ON WHEN Stage 3 of 7

- [ ] Documented evaluation results that meet (or clearly miss) the Stage 1 success criteria
- [ ] An explicit, evidence-based go/no-go decision on whether to build for production

3.4 Stage 4 - Build for Production

1 2 3 **4** 5 6 7

Lifecycle · stage 4 of 7

A prototype proves the task can be done; this stage turns it into a system you can run reliably, securely and economically every day.

A working prototype is not a finished product. A prototype that performs well in a demo can look almost finished, and it is easy for leaders to conclude the job is nearly done when it isn't. As a rule of thumb, the prototype is around 20% of the effort. The other 80% is Stages 4 to 6: the security, reliability, cost controls, human review screens, testing, rollout and change management that make it safe to run every day. Make the move to production a deliberate decision.

Be equally wary of a prototype whose status is unclear. One that drifts into everyday use without a clear go/no-go decision is running in production without production safeguards. Label prototypes clearly and set expectations about the remaining effort early.

Start slow and layer on capability. Get one small workflow working, then extend it. For example, an agent might first investigate a reported fault and write a ticket summarising what it thinks went wrong; only once that is reliable would you let it propose and make the fix.

- **Build in reliability and security** using the controls in Section 4, and decide how this agent fits within your wider governance approach.
- **Control cost** using the levers in Section 5.
- **Build the human touchpoints**: the approval, review and escalation screens appropriate to the autonomy level, designed with the people who will use them.
- **Use your existing systems.** Avoid large-scale data migrations. Expose existing data (SharePoint, databases, ERP) through interfaces and let the agent orchestrate across them. This saves time and money and avoids ripping out systems the organisation has invested years and significant sums in.
- **Decide where the agent will run** (below).

Where should the agent run?

This is one of the most consequential architecture decisions, but the principle is simple: **default to a managed agent platform** that meets your security and compliance requirements, and move toward running

it yourself only where a specific requirement (data residency, isolation, latency or cost at scale) justifies the extra operational burden. Control and operational effort rise together, so don't buy control you don't need.

	Managed agent platform	Your own cloud	On-premises
Operational effort	Lowest	High	Highest
Control over data and network	Lower to medium	High	Highest
Reaching systems behind your firewall	Hardest	Easier	Easiest
Time to production	Fastest	Slower	Slowest
Best for	Speed, low overhead	Compliance and control	Air-gap or sovereignty requirements

Among managed agent platforms, the major cloud providers (Microsoft, AWS and Google) and the AI model providers all offer hosted agent environments, broadly similar to operate. The deciding factor is usually **ecosystem fit**: which cloud your data, identity and governance already live in. Leading models are available across several platforms, so the choice of model and the choice of platform are largely independent. These platforms are evolving rapidly, so check current capabilities, availability and pricing before deciding.

READY TO MOVE ON WHEN Stage 4 of 7

- ☐ Security, audit, reliability and cost controls in place
- ☐ A documented and implemented hosting decision
- ☐ Human touchpoints that match the autonomy level

3.5 Stage 5 - Test

1 2 3 4 **5** 6 7

Lifecycle · stage 5 of 7

Before real users or real data are involved, confirm the agent meets its quality, safety and cost thresholds under realistic conditions. Scale the depth of testing to the risk and impact of the agent.

- **Quality.** Run the full eval set against the exact production configuration. How often does the agent make things up ("hallucinate"), and what is the impact when it does?
- **Safety and security.** Deliberately try to misuse the agent, including hiding malicious instructions in the content it reads, and confirm the guardrails and approval points hold.
- **Failure handling.** Simulate system outages and bad inputs, and confirm the agent fails safely and escalates correctly.

- **Cost and speed.** Measure cost and turnaround time per task, for typical cases *and* worst cases, and confirm cost limits trigger.
- **Volume.** Validate behaviour and cost at expected production load.
- **User acceptance.** The business owner and end users validate real outputs and the review and approval experience.

What makes a good evaluation. A good eval checks more than whether the final answer is right; it checks that the agent reaches it the way a capable person would. Trace the steps an expert takes and check the agent at each one. Start small: sit with an expert to define what an excellent outcome looks like for a handful of representative cases, then hold the agent to that standard. Rigorous evaluation also builds the business case. Many stakeholders remain sceptical that agents work, and solid evidence is often what a leadership team needs to trust that an agent will deliver a return.

FOR YOUR TECH TEAM

Automate the eval suite and run it in your CI/CD pipeline so every change affecting model behaviour is tested quickly and safely. Red-team specifically for prompt injection via untrusted content and tool misuse. Inject tool timeouts, API errors and malformed inputs, and confirm graceful degradation with no state corruption. Load-test at expected concurrency.

READY TO MOVE ON WHEN Stage 5 of 7

- [ ] A test report showing the agent meets its quality, safety, reliability and cost thresholds
- [ ] A documented go/no-go decision signed off by the owners

3.6 Stage 6 - Deploy

1 2 3 4 5 **6** 7

Lifecycle · stage 6 of 7

Release to real users in a controlled, reversible way, with the same rigour as any production IT system and extra caution because of the agent's autonomy.

- **Launch at low autonomy** (L1 or L2) even if the target is higher.
- **Roll out gradually**: a pilot group first, or a "shadow mode" in which the agent runs alongside people without acting, before full rollout.
- **Have a kill switch.** You must be able to switch the agent off and revert to the previous process quickly.
- **Make sure monitoring is live** before real traffic arrives, not after.
- **Prepare people.** Communicate what the agent does and why, train teams to review its work and escalate issues, and provide supporting documentation (see Section 6).

READY TO MOVE ON WHEN Stage 6 of 7

- [ ] The agent is live for its initial group at a safe autonomy level
- [ ] Monitoring, alerting, audit logging and a tested kill switch all working

3.7 Stage 7 - Maintain

1 2 3 4 5 6 **7**

Lifecycle · stage 7 of 7

Agents are living systems. Their inputs, the underlying AI models and the surrounding business can all change, and behaviour drifts if left unattended.

- **Monitor continuously.** Track success rate, cost per task, turnaround time, errors and escalations against the baseline, and alert on drift.
- **Re-test on every change.** Treat the eval set as a permanent quality gate. Re-run it before adopting a new model version, prompt change or tool update; model upgrades can silently change behaviour.
- **Review autonomy periodically.** Raise it for workflows that are performing well, and lower it if reliability slips.
- **Manage cost** against the cost model and investigate drift.
- **Audit regularly.** Review logs, permissions and data access, and keep the agent register current.
- **Be ready for incidents.** Keep a runbook covering how to detect, contain (kill switch), investigate and fix problems, with a named owner.
- **Gather user feedback** and feed it into improvements.
- **Retire deliberately.** When an agent is superseded, revoke its access, archive its logs and remove it from the register.

STEADY STATE Stage 7 of 7

- [ ] Live monitoring against baselines
- [ ] An eval suite gating every change
- [ ] An active cost model
- [ ] Current audit and register records
- [ ] A tested incident runbook with a named owner

SECTION 04

Governance

Governance is what separates a demo from a system you can run confidently. An agent that can take actions is, in effect, software that decides for itself which commands to run and which data to touch. A useful mental model is to treat it as a new, inexperienced member of staff with system access: it needs its own identity, only the access the job requires, supervision proportionate to the risk, and a record of what it did.

4.1 Security

No single control is enough. Security is about layers, so that no single failure opens a straight path to harm. This is often described as a "Swiss cheese" model, in which each layer has holes but the holes don't line up.

- **Give the agent its own identity and minimal access.** Never let it borrow a person's broad credentials, and assume zero trust.
- **Expose only the tools it needs.** Every extra tool widens the scope for misuse and mistakes.
- **Require human approval for irreversible or sensitive actions**, such as moving money, deleting data, changing permissions or publishing externally, even when the agent is otherwise autonomous. Alternatively, don't allow them at all.
- **Run it in an isolated environment** with tightly restricted access to networks and files.
- **Treat everything the agent reads as data, not instructions.** Agents that read emails, documents, web pages or tickets can be targeted by "prompt injection", where malicious instructions are hidden in content to hijack the agent.

FOR YOUR TECH TEAM

Security baseline: a dedicated, least-privilege identity; short-lived scoped credentials with secrets in a vault, never in prompts or logs; sandboxed execution (container, VM or managed sandbox) with default-deny network egress and an allow-list of required endpoints; approval gates on all irreversible actions; validation of tool inputs; separation of trusted system instructions from untrusted content; monitoring for anomalous tool-call patterns; and redaction of sensitive data from logs and traces.

4.2 Auditability

After the fact, you must be able to answer: *what did the agent do, why, using what data, and on whose authority?* This is essential for investigating problems, compliance and trust.

- **Record every step**: inputs, the agent's decisions, every action and its result, and the final output.
- **Make outcomes explainable and reproducible** by recording which model, instructions and configuration were used for each run.
- **Record who or what started each run**, and keep logs tamper-evident, retained in line with your compliance regime.
- **Keep sensitive data out of logs.** Audit logs easily become a new source of data leaks.

FOR YOUR TECH TEAM

Use a structured tracing standard so traces are queryable. Many agent frameworks don't provide production-grade logging out of the box, so plan for it. Version system prompts, tools and configuration alongside model versions. Redact or tokenise PII and secrets before anything is written.

4.3 Reliability

Agents are non-deterministic and depend on external systems, so reliability has to be designed in.

- **Set hard limits** on steps, time and cost per task, with a safe stop when a limit is hit.
- **Handle failures gracefully.** Systems time out and fail, so a temporary failure must not corrupt data or leave work half-done.
- **Escalate rather than guess.** When the agent is uncertain or hits an exception, it should hand off to a person. Guard explicitly against over-confidence, and design the hand-off deliberately.
- **Keep testing continuously** against the eval set (Stage 7).

FOR YOUR TECH TEAM

Bound iterations, wall-clock time and token budget per run. Build retries with backoff, fallbacks and clear error handling. Make side-effecting tools idempotent (for example, idempotency keys on writes) so a retry never duplicates an action.

4.4 Organisational governance

- **Ownership.** Every production agent has a named owner accountable for its behaviour, cost and incidents.
- **Approval.** New agents and significant changes pass a review of security, data access and potential impact before going live.
- **A register.** Keep a central record of every agent: its purpose, data access, autonomy level and owner. You can't govern what you haven't catalogued.
- **Compliance.** Map each agent's data handling to your regulatory obligations (for example, data residency and sector rules) and to your AI provider's terms.

SECTION 05

Cost

Agents have a cost profile that surprises teams used to conventional software or single AI calls. Because an agent loops (reasoning, acting, checking and reasoning again), one task can involve many AI calls. Cost must therefore be modelled **per completed task**, not per call, and monitored continuously.

Where the cost comes from. AI model usage is usually the largest element, and it grows with the number of steps and the amount of context the agent carries. Runaway loops are a real budget risk. Other costs include the infrastructure (secure environments, external services, logging), the reviewer time needed at lower autonomy levels, and the engineering effort to build, test and maintain the agent.

The main levers:

- **Right-size the model.** Use a highly capable model where reasoning is hard and cheaper, faster models for simpler steps.
- **Cap the cost of every task**, which limits both runaway loops and worst-case bills.
- **Invest in good verification.** It reduces wasted attempts and re-runs, often the biggest hidden cost.
- **Batch work where speed isn't critical**, as this is materially cheaper than real-time processing.
- **Raise autonomy as confidence grows**, reducing reviewer time.

Build a cost model before going to production. Estimate typical and worst-case cost per task, add infrastructure and human review, and compare the total with the fully loaded cost of the current process. Track actual spend against the model from day one, because costs drift as inputs, instructions and models change.

Budget for the worst case, not the average. An agent that usually finishes in three steps but occasionally loops twenty times will break a budget set on the average. Hard per-task limits turn an open-ended risk into a known, capped cost.

FOR YOUR TECH TEAM

Use prompt caching for stable prefixes (system instructions, tool definitions, reference context); it is one of the highest-leverage savings for looping agents. Compact or summarise context on long runs, and retrieve on demand rather than pre-loading. Use batch APIs for non-interactive, high-volume work. Route steps across models of different sizes. Right-size infrastructure and sample logs where appropriate.

SECTION 06

People and Change

The technology is rarely the hardest part. Agentic AI changes how work is done, and often who does it, so the classic disciplines of digital transformation and change management matter at least as much as for any other technology programme. Arguably they matter more, because agents can reshape job roles and responsibilities, which needs more sensitive handling than a typical system rollout.

- **Involve end users from the start.** The people who do the work today know where the real complexity, exceptions and value lie. Involve them in selecting the use case (Stage 1), mapping the process (Stage 2), designing the review and approval experience (Stage 4) and acceptance testing (Stage 5).
- **Communicate early, clearly and honestly**: what the agent will and won't do, why the change is being made, and what it might mean for people's roles. Silence breeds anxiety and resistance.
- **Address the impact on roles directly.** Where responsibilities change, plan the transition deliberately with HR and line managers.
- **Train people for their new role alongside the agent**: how to review its work, when to override it, and how to escalate. Provide clear support routes.
- **Secure visible sponsorship.** A named business owner and leadership support signal that this is a business change, not an IT experiment.
- **Measure adoption, not just performance.** An accurate agent that people work around has delivered nothing. Track usage and satisfaction, and act on feedback.

The organisation's wider AI awareness and culture are also key ingredients of success. The training and other methods to support this foundation are a whole other topic: they're out of scope here, but recognise that you're unlikely to be successful if you don't give them some attention first.

FROM MY EXPERIENCE

I prototyped a market research agent in a niche domain for use by a client's sales personnel to find new target customers. The prototype gave a working example of their concept and showed clear potential to massively speed up the market research. However, when the sales team was engaged, it transpired that how they worked and what they needed were different from what the prototype assumed.

SECTION 07

Adoption Roadmap

For an organisation starting its agentic AI journey, the recommended path is deliberately incremental: build capability and trust on safe workflows before expanding scope and autonomy.

1. **Foundations.** Put the basics in place: access to AI models, a shared agent framework, logging and monitoring, an identity and access model for agents, and the agent register.
2. **First pilot.** Pick one boring, high-volume, verifiable, internal workflow and take it through the full lifecycle at low autonomy.
3. **Operationalise.** Stabilise the pilot, prove the cost model and raise autonomy as evidence allows. Capture reusable components and lessons, including on adoption and change.
4. **Scale out.** Apply the same lifecycle to further workflows, reusing the shared platform, governance and evaluation tooling.
5. **Expand scope.** Only then take on higher-stakes or customer-facing workflows, with the controls and operational maturity earned along the way.

GUIDING PRINCIPLE

Start where the work is verifiable and the impact of mistakes is small, and turn up autonomy as evidence, not enthusiasm, accumulates.

Andy Hickman
ADVISORY

ABOUT THE AUTHOR

Andy Hickman is a strategic technology adviser based in Bristol. Through Andy Hickman Advisory he helps organisations with digital strategy, AI adoption and governance, technology risk and technical due diligence. He is also AI Adviser in Residence at SETsquared Bristol, supporting startups with their use of AI.

EMAIL
andy@andy-hickman.co.uk

WEB
andy-hickman.co.uk