PLAN · BUILD · TEST · RELEASE · OPERATE

Agent principles & *standards*.

A living, open-source set of principles for building, evaluating, releasing and operating AI agents over the long term.

> *An agent is software. The disciplines that have kept ordinary software reliable for decades apply directly. None of this is new. It is just applied to a new kind of system.*

Living document · **Open** source · Vendor & model **agnostic** · CC BY 4.0

REFERENCE

Contents

INTRODUCTION

An agent estate is cheap to build and expensive to own.

Organisations are increasingly building, buying and running AI agents: software that uses an AI model to take actions on someone's behalf. Agents can be powerful in the right situations, but they are unusual. They are shaped by written instructions rather than only code, they change when the underlying model changes, and they are not perfectly predictable.

Left unmanaged, an agent estate becomes expensive to own. Instructions drift, a small change to a tool quietly breaks behaviour, nobody is quite sure why the agent did what it did, and the same capability gets rebuilt three times. The cost of owning the agent over its lifetime, not the cost of building it, is where the money goes.

The core idea is simple: an agent is software, and the disciplines that have kept ordinary software reliable for decades apply directly. Keep track of every version of everything. Check it works automatically, not just by hand. Release changes carefully and be able to undo them. Watch it closely once it is live.

These principles apply to all new agents and to the modernisation of existing ones, whether built in-house, supplied by a vendor, or assembled from off-the-shelf parts. The origin does not matter; this document uses **supplier** to mean whoever provides it. An agent that cannot meet a principle should be flagged early, during planning. A principle may be waived where there is a clear, demonstrable benefit, but the waiver should be written down, time-limited, and revisited.

WHO THIS IS FOR

Not just engineers.

This document is for anyone involved in commissioning, building, approving, or operating an AI agent. Product owners, operations staff, risk and compliance teams, and leadership should all be able to read, understand, and apply these standards.

The guidelines are deliberately vendor, model, and framework-agnostic. Specific tools evolve rapidly; these core engineering principles endure.

OUT OF SCOPE

This document focuses strictly on technical and operational standards. It does not address organisational enablement, such as target operating models, skills development, or enterprise education programmes.

HOW TO READ THIS DOCUMENT

Three layers, read to the depth you need.

Every section is built in three layers. A non-technical reader can read the first two and come away with the full picture. A practitioner may want the third. Technical terms are written in their precise form rather than watered down, and every such term is defined in the glossary.

1

The principle

One sentence. The thing to remember.

2

Why this matters

A short, plain explanation of what goes wrong without it.

3

In practice

The specific standards. More detailed, more technical. Read this when you are doing the work.

THE WORDS WE USE

MUST, SHOULD, MAY.

The standards use three levels of obligation, following the long-established convention of RFC 2119. In plain terms:

MUST **MUST NOT**

Non-negotiable. If you cannot meet this, you need an explicit, written waiver.

SHOULD **SHOULD NOT**

Strongly expected. You can deviate, but you need a good reason and you should write it down.

MAY

Genuinely optional. A choice, not a requirement.

FOUNDATIONS

Two constraints you carry throughout.

These are not a stage you pass through. They apply at every stage of an agent's life.

Ethics & responsible use

THE PRINCIPLE

An agent acts on behalf of people, so someone must always be accountable for what it does, and it must never be allowed to do more than it was meant to.

WHY THIS MATTERS

Agents take actions in the real world. They send messages, move money, change records, influence decisions. When something goes wrong, "the model did it" is not an answer anyone can act on. Responsibility has to sit with a person. And because an agent will do whatever its instructions and tools allow, the safest agent is one whose permissions are deliberately kept narrow.

IN PRACTICE

- Every agent **MUST** have a documented purpose: who it serves, what it is allowed to do on their behalf, and just as important, what it is not for. Most ethical failures start as scope creep.
- Every agent **MUST** have a named human owner who is accountable for its behaviour. Accountability **MUST NOT** be diffused across "the model" or "the platform".
- Agents **MUST** be transparent about being agents. Pretending to be a human **MUST NOT** be a default behaviour.
- High-impact, irreversible or sensitive actions (moving money, sending communications, deleting data, decisions that affect a person) **MUST** require explicit human confirmation, unless a documented and reviewed policy expressly allows otherwise.
- Any agent that influences decisions about people **MUST** be assessed for bias and unfair impact. That assessment **MUST** be repeated whenever the model, the instructions, or the data sources change.
- Ways the agent could be misused **MUST** be considered during design. For any agent exposed outside the organisation, adversarial testing **SHOULD** be a standard part of evaluation.
- Agents **MUST** refuse actions that break the organisation's published policies, the law, or the rights of others, and that refusal behaviour **MUST** itself be tested.
- There **MUST** be a clear route for a person to question, correct or appeal what an agent has done, especially where it affects them directly.
- The energy and compute cost of running an agent **SHOULD** be a conscious consideration, particularly for high-volume or long-running agents.

Data

THE PRINCIPLE

Know what data the agent can see, where that data goes, and keep both as small as possible.

WHY THIS MATTERS

Agents are unusually data-hungry. In a single task an agent can read a document, query a database, send information to a model provider, and write a record somewhere else, often crossing organisational and trust

boundaries as it goes. If you cannot answer "what does this agent know, and where does it go?", you cannot keep it safe or compliant.

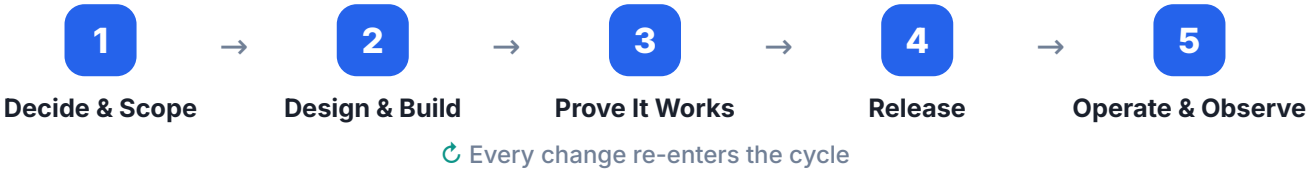
IN PRACTICE

- The data an agent reads, writes, sends and keeps **MUST** be documented as part of its design, before it ships.
- Data minimisation applies: an agent **MUST** be given the smallest set of data and the narrowest tool access it needs. Broad credentials and wildcard permissions are not acceptable defaults.
- Personal data, credentials and other sensitive information **MUST** be encrypted in transit and at rest, and handled in line with applicable data protection law.
- Personal or sensitive data **MUST NOT** be placed into prompts, evaluation fixtures, or logs unless there is a documented lawful basis and a matching retention and access policy.
- All conversation and trace data **MUST** have a defined retention period. Keeping it forever is not acceptable.
- Where a third-party model provider is used, the contract **MUST** be documented: whether inputs train the provider's models, how long data is kept, and who else processes it. If the default is to train on inputs, that **MUST** be switched off unless consciously chosen otherwise.
- The provenance of data the agent relies on (source, freshness, ownership) **SHOULD** be tracked and surfaced to the user where relevant. Citations are an integrity mechanism, not a decoration.
- Anything the agent produces **MUST** be traceable back to the agent version, the instruction version, and the model version that produced it.
- Data the agent generates artificially **MUST** be marked as such, so it is not mistaken for real data later.
- For any agent serving multiple tenants, leakage of one tenant's data to another **MUST** be explicitly tested for.

THE AGENT LIFECYCLE

Five stages, then it loops.

The rest of this document follows an agent through its life as a product, from the decision to build one to running it day to day. Most agents move through these five stages in order, then loop; every change re-enters the cycle.



1 · Decide and scope (Plan)

THE PRINCIPLE

Decide deliberately whether something should be an agent at all, and define exactly what it is for before building it.

WHY THIS MATTERS

The most expensive agent is the one that should never have been built, or the one whose job was never clearly defined. An agent with a fuzzy remit grows uncontrollably, becomes impossible to test, and accumulates risk. Clear scope at the start is the cheapest quality control there is.

IN PRACTICE

- The problem the agent solves MUST be defined before the agent is designed, including what success looks like and how it will be measured.
- Existing agents, skills and tools SHOULD be favoured over building something new, where they meet the need. Reusing existing parts is far cheaper to own.
- An agent SHOULD do one thing well. A single agent that tries to do everything is harder to test, change, and reason about than several focused ones working together.
- The choice to build an agent, rather than ordinary software or no automation at all, SHOULD be a conscious, recorded decision. Agents are the right tool when a task needs judgement over ambiguous inputs; they are the wrong tool when a task simply needs to be correct every time.
- The intended audience and the boundaries of acceptable use MUST be defined at this stage, because they shape everything that follows.
- Risky or behaviourally complex requirements MUST be identified now and tackled early in the build, not deferred.

2 · Design and build

THE PRINCIPLE

Treat every part of the agent (its instructions, skills, tool definitions, configuration) as a versioned, reviewed asset, just like code.

WHY THIS MATTERS

An agent's behaviour lives in more than its code. It lives in the system prompt, in each skill, in the wording of every tool description. If those things exist only in someone's head or a vendor's web console, they cannot be

reviewed, compared, rolled back, or trusted. The single biggest driver of long-term cost is agent behaviour that nobody can see or control.

IN PRACTICE

- All agent assets (system prompts, skills, tool definitions, configuration, evaluation suites) **MUST** be held in version control. An instruction that exists only in a vendor console is not under engineering control.
- Every change to any of these assets **MUST** be peer reviewed before it reaches production. A change to a prompt is a change to behaviour and carries the same risk as a change to code.
- Agents **SHOULD** be decomposed into the smallest coherent units of capability, with clear single responsibilities, rather than one large set of instructions.
- Where an action must reliably be correct (a calculation, a record update, a data transformation) it **SHOULD** be performed by a tool, not left to the model. Use the model for judgement; use code for correctness.
- System prompts, skills and tool descriptions **MUST NOT** contain credentials, customer data, or environment-specific values. Those **MUST** be supplied through configuration.
- Tool and skill descriptions **MUST** be written clearly for the agent that will read them: unambiguous about when to use the tool and what each input means. The description is part of the contract.
- Shared instruction content **SHOUL**D be composed from named, reusable fragments rather than copied between agents, so it does not drift apart over time.
- There **MUST** be an automated way to build, test and package the agent, independent of any individual's machine or any vendor console.
- A reproducible environment that lets a contributor run and test the agent **SHOULD** be available.
- Each agent **SHOULD** have a documented set of expected failure modes and the intended behaviour for each. Failures should be designed for, not discovered.

3 · Prove it works (Test)

THE PRINCIPLE

An agent's behaviour MUST be demonstrated through evaluated, repeatable, recorded checks before and after every change.

WHY THIS MATTERS

You cannot tell whether an agent works by trying it once. Its behaviour is not perfectly predictable, it changes when the model or instructions change, and it can fail in ways ordinary software never does. Evaluation is to an agent what testing is to ordinary software. An agent without an evaluation suite is an agent nobody can safely change.

IN PRACTICE

- Every capability a user would recognise MUST have at least one evaluation case.
- Evaluations MUST record a human-readable transcript: the input, the steps the agent took (including every tool call), the output, and the judgement made. A failed evaluation must be diagnosable without re-running it.
- Evaluations SHOULD exercise the agent the same way a real user or system would, and SHOULD run against real tools and real model endpoints wherever feasible. Where that is not possible, a representative subset MUST run end-to-end, and the approach for the rest MUST be documented.
- Evaluation suites MUST be held in version control alongside the agent, and MUST run automatically whenever an asset changes, not only by hand and not only at release time.
- Exploratory testing by humans remains essential. Anything found by exploration MUST be turned into an automated evaluation case where possible, so the same failure cannot silently come back.
- Evaluation assets MUST be held to the same quality standard as the agent itself.

Evaluation should be layered, so feedback is both fast and thorough:

- **Component checks** — individual skills or prompts tested in isolation against fixtures. Fast, and safe to run on every change.
- **End-to-end checks** — full, user-recognisable tasks, tested through the real interface with real tools.
- **Adversarial & safety checks** — for any externally exposed agent, a suite covering prompt injection, jailbreak attempts, data exfiltration and misuse. This suite MUST be kept up to date as new attack patterns emerge.
- **Performance & cost checks** — latency and cost-per-task measured against an agreed profile. A regression in cost is a regression.

4 · Release / deploy

THE PRINCIPLE

Release agent changes the way you would release changes to critical software: as versioned, tested bundles, rolled out carefully, and reversible within minutes.

WHY THIS MATTERS

Because an agent's behaviour can shift with a one-word change to a prompt, releasing carelessly is how a small edit becomes a production incident. The protections are the same ones critical software has relied on for years: know exactly what you are releasing, prove it in a safe environment first, roll it out gradually, and be able to undo it instantly.

IN PRACTICE

- Every agent asset **MUST** be versioned: prompts, skills, tool definitions, configuration, and the model version itself. A release is a versioned bundle, not an edit made directly in a console.
- The model version used in production **MUST** be pinned. Automatically picking up "the latest model" without an evaluation gate is not acceptable. An upstream model change is a behaviour change and **MUST** be evaluated before adoption.
- Separate environments for testing and for production **MUST** exist, and behaviour **MUST** be evaluated in a testing environment before being promoted.
- It **MUST** be possible to roll any change back quickly, in minutes, not days.
- Significant instruction or model changes **SHOULD** be rolled out gradually to a small share of traffic first, or compared side by side against the current version, with evaluation results deciding whether the rollout continues.
- Suppliers **MUST NOT** have direct access to production model endpoints, tool credentials, or production data. Releases are carried out through automated, repeatable processes, not by hand.
- Configuration changes **MUST** go through the same review, testing and release path as changes to code or prompts. A configuration change is a behaviour change.
- The ability to switch individual tools or capabilities on or off without a full redeployment **SHOUL**D be available.

5 · Operate and observe (Monitor)

THE PRINCIPLE

Once an agent is live, you MUST be able to see what it is doing, prove it is still working, control who and what it can reach, and contain its costs.

WHY THIS MATTERS

An agent that is live but unobservable is the most expensive thing in this document. When something goes wrong (and with a non-deterministic system, something will), the difference between a five-minute fix and a five-day investigation is whether you recorded what the agent actually did. Agents fail in quiet ways: not a crash, but a slow drift in quality, a creeping cost, a rising refusal rate. Those are only caught by watching for them deliberately.

IN PRACTICE

Seeing what it does — observability

- Every agent's turn **MUST** be traceable. Records **MUST** capture, at minimum: the input, the model and instruction versions used, every tool call with its inputs and results, token usage, latency, and the final output.

- Logs **MUST** be structured, use a consistent timestamp standard, and flow into the organisation's standard logging systems.
- Personal, sensitive or confidential data **MUST NOT** be written to logs in readable form. Where the input itself contains such data, it **MUST** be redacted before the record is stored.
- Errors and warnings **MUST** be actionable; a log entry should make clear what to investigate.
- It **MUST** be possible to change how much detail is logged without redeploying the agent.

Proving it still works — monitoring & continuous evaluation

- Each agent **MUST** expose a health check that reflects the true state of its dependencies (model provider reachable, tools available, credentials accessible) and is safe to check frequently.
- Live behaviour **MUST** be monitored with measures suited to agents: error rate, refusal rate, escalation rate, latency, cost per task, and evaluation scores on a sample of real traffic.
- Drift, a gradual change in any of those measures away from the established baseline, **MUST** trigger investigation. A silent quality drop is the most expensive kind of incident.
- Alerts **MUST** be defined for material regressions in those measures.
- A sample of real production activity **SHOULD** be fed back into the evaluation suite. New failures seen in production **MUST** be added so they cannot silently recur.

Controlling what it can reach — access & cost

- Access to the agent, and to every system it can call, **MUST** be authenticated and authorised through the organisation's standard identity systems.
- Tool permissions **MUST** be scoped per agent and, where it applies, per role and per user. An agent **MUST NOT** hold broader permissions than its job requires.
- Every tool action an agent takes **MUST** be recorded in an audit log, with enough context to attribute it to the originating request, user, and agent version.
- High-impact tools **MUST** carry additional controls (confirmation, approval steps, or stronger authentication) in proportion to their impact.
- Cost ceilings (per request, per user, per day) **MUST** be configurable and enforced. Runaway loops are a known, recurring failure mode.
- Long-running or repeating agent behaviour **MUST** have hard stopping conditions: a maximum number of steps, a maximum token budget, or a maximum running time.
- Agents **MUST** handle the failure of the things they depend on gracefully, using sensible retries, fallbacks, and clear surfacing of failure. Silent failure is the worst outcome for an agent.

WORKING WITH SUPPLIERS

Assume the ground will shift.

THE PRINCIPLE

Design and operate as though every supplier-provided component, the model especially, will change with limited notice or consent, because in time it will.

WHY THIS MATTERS

Most agent strategies are built on a quiet assumption that the model underneath will behave roughly the same tomorrow as it does today. That assumption is not safe. The model is rarely owned by the organisation deploying the agent; it is licensed, accessed through an API, or embedded inside a tool, and in every configuration the supplier retains the right to change it. Models are deprecated. Versions are retired. Behaviour drifts as providers retrain, retune, or adjust safety filters. Output formats shift in ways small enough to slip past a glance and large enough to break a downstream parser.

Contractual assurances about stability or advance notice reduce surprise and create recourse, but they do not give the organisation control. An agent built on a third-party model is dependent on infrastructure that someone else operates and someone else evolves, much as you are with cloud services. A supplier's failure to evaluate, observe, version, or control their part of the system is, in operational terms, your failure.

IN PRACTICE

- Suppliers **MUST** be told that these principles apply, and any area where their agent cannot meet them **MUST** be surfaced during selection, not after.
- Suppliers **MUST** provide the assets needed to evaluate, operate and observe the agent to the standards above, including the ability to inspect its behaviour and trace what it does.
- A supplier's proposed changes, including upstream changes to a model the agent depends on, **MUST** be evaluated before adoption, the same as any internal change.
- Open standards and components that do one thing well **SHOULD** be favoured over tightly bundled, proprietary suites. The former can be composed and replaced; the latter leads to lock-in.
- Where a supplier operates part of the agent on the organisation's behalf, that part **MUST** be held to the same standards as anything operated in-house.

GLOSSARY

The words, defined.

Agent
Software that uses an AI model to pursue a goal, typically by taking a sequence of actions and using tools, rather than producing a single response.

Foundation model
The underlying AI model an agent is built on. The same agent can behave differently on different models or versions.

Model version pinning
Deliberately fixing the agent to a specific model version, so its behaviour does not change unexpectedly on a provider update.

System prompt
The standing set of instructions that defines an agent's role, behaviour and boundaries. A behaviour-shaping asset, treated like code.

Prompt
Any instruction given to the model, including the system prompt and the wording of individual requests.

Skill
A self-contained, reusable unit of capability or instruction an agent can draw on. Treated as a versioned asset.

Tool
A defined capability the agent can call: query a database, send a message, perform a calculation. The agent decides when to use it from its description.

Tool definition / schema
The machine-readable description of a tool: what it does, what inputs it takes, what it returns. Its wording is part of the agent's behaviour.

MCP
Model Context Protocol. An open standard for exposing tools and data to agents consistently, so components from different sources can work together.

Token
The unit in which AI models measure and price text. Token usage is the main driver of an agent's running cost.

Context window
The amount of information (in tokens) a model can consider at once. A finite resource that has to be managed.

Evaluation (eval)
A repeatable, recorded check of an agent's behaviour against an expected outcome. The agent equivalent of a test.

Evaluation suite
The full collection of evaluations for an agent, run automatically when assets change and at release time.

Fixture
A fixed, known piece of input data used to test a component in isolation, so the result is repeatable.

Trace
The full record of a single agent turn: input, every step and tool call, token usage, timings, and output.

Prompt injection
An attack where hostile instructions are smuggled into content the agent reads, to make it act against its purpose.

Red-teaming
Adversarial testing. Deliberately trying to make an agent misbehave, to find and fix weaknesses before someone else does.

Drift
A gradual, often unnoticed change in an agent's behaviour or performance over time, away from its baseline.

Rollback
Reverting an agent to a previous known-good version quickly, to contain a problem introduced by a change.

Fallback
A defined alternative behaviour for when something the agent depends on fails.

Supplier
Whoever provides an agent or a component of it: an internal team or an external vendor.

Version control
A system that records every version of every asset, who changed it and why, and allows any version to be restored.

CONTRIBUTING & LICENCE

This document is meant to evolve.

Contributions, corrections and challenges are welcome via pull request or issue. Where you disagree with a principle, open an issue with your reasoning. Disagreement is how the document improves.

Kieran Cornwall / The Honest Generalist

Free to share, it started here —
kierancornwall.com