Agentic AI Architecture Checklist

A practical guide for defining, scoping, and preparing enterprise-grade AI agents

1. Introduction

Agentic AI refers to artificial intelligence systems that can pursue goals, reason through tasks, use tools, interact with systems, and take actions with varying levels of autonomy. Unlike a traditional chatbot that mainly responds to user prompts, an AI agent can plan a sequence of steps, gather information, call APIs, update records, escalate exceptions, and monitor outcomes.

This checklist is designed to help business leaders, architects, product managers, AI engineers, compliance teams, and operations teams think through the key design decisions before building or deploying an agentic AI solution. It focuses on practical architecture questions: what problem the agent solves, what decisions it can make, what systems it can access, how success will be measured, and what controls are needed before production use.

1.1 Why Agentic AI Architecture Matters

Agentic AI architecture matters because agents are not just content-generation tools. They operate as goal-driven systems that may access enterprise data, select tools, execute tasks, and influence business outcomes. A poorly designed agent can create operational risk, compliance exposure, inaccurate decisions, unnecessary cost, or user frustration. A well-designed agent, on the other hand, can improve productivity, reduce cycle time, standardize decision support, and create a more responsive digital operating model.

- **Autonomy requires boundaries:** The more an agent can decide or execute independently, the clearer its limits must be.
- **Tool access creates real-world impact:** If an agent can update a CRM, send an email, create a ticket, or change a workflow status, architecture must include permissions, validation, and rollback paths.
- **Context quality affects output quality:** Agents depend on accurate data, current policies, trusted knowledge sources, and reliable memory.
- **Governance must be designed early:** Audit trails, human approvals, escalation rules, monitoring, and security controls should not be added as an afterthought.
- **Production reliability is different from a demo:** A prototype may work in a simple scenario, but production agents must handle exceptions, ambiguous inputs, missing data, latency, and changing business rules.

Example: A customer support agent that only drafts suggested replies is low risk because a human reviews the response before sending. However, an agent that can automatically issue refunds, update customer records, or close complaints has higher risk. Its architecture must include approval thresholds, transaction limits, identity-based access, logging, and exception handling.

1.2 How to Use This Checklist

Use this checklist as a planning and review tool before starting an agentic AI build, during solution design workshops, and before moving a pilot into production. Each

section can be used as a discussion guide with stakeholders from business, technology, data, security, legal, compliance, risk, and operations.

- **For business teams:** Use it to clarify the business problem, value case, users, and success metrics.
- **For architects and engineers:** Use it to define agent components, system integrations, memory, orchestration, tool access, and monitoring needs.
- **For governance and compliance teams:** Use it to identify risk controls, approval points, audit requirements, and responsible owners.
- **For product owners:** Use it to convert a broad idea into a scoped backlog with clear requirements and acceptance criteria.

A practical way to apply this checklist is to score each item as **Ready**, **Partially Ready**, or **Not Ready**. Items marked Partially Ready or Not Ready should become design actions, risk items, or backlog tasks before the agent is approved for wider use.

1.3 What You'll Learn

By the end of this checklist, you will have a structured understanding of how to define an agentic AI use case before making technology decisions. You will know how to describe the business problem, define the agent's purpose, limit its operating scope, identify required users and systems, and select meaningful success metrics.

- How to distinguish a real agentic AI use case from a simple automation or chatbot use case.

- How to define an agent's goals, responsibilities, and boundaries.
- How to identify the people, data sources, tools, and systems the agent must interact with.
- How to choose success metrics that measure business value, operational performance, user experience, and risk control.
- How to document examples and assumptions so that stakeholders share the same understanding of what the agent will and will not do.

2. Define Your Agentic AI Use Case

The first architecture decision is not which model to use. The first decision is whether the use case truly needs an agent. Many business needs can be solved with a workflow, rules engine, dashboard, search experience, or standard automation. Agentic AI is most valuable when the task requires reasoning over context, making choices among tools, handling multi-step workflows, adapting to changing information, and coordinating actions across systems.

Use this section to convert a broad idea such as “build an AI operations assistant” into a well-defined use case such as “an AI agent that reviews incoming support tickets, classifies urgency, checks the knowledge base, drafts a response, recommends escalation, and creates a follow-up task when required.”

2.1 Identify the Business Problem

Start by documenting the business problem in simple operational language. Avoid beginning with a technology statement such as “we need an AI agent.” Instead, describe the pain point, who experiences it, how often it occurs, what it costs, and why the current process is insufficient.

- **Problem statement:** What specific issue should the agent help solve?
- **Current process:** How is the work performed today, and where do delays or errors occur?
- **Business impact:** What are the measurable consequences, such as cost, time, risk, missed revenue, poor experience, or compliance exposure?

- **Frequency and volume:** How often does the problem occur, and how many cases, requests, records, or transactions are involved?
- **Root causes:** Is the problem caused by manual effort, fragmented systems, inconsistent judgment, lack of visibility, slow decision-making, or poor data access?

Example 1: Support operations. A company receives thousands of support tickets each month. Human agents spend significant time reading ticket descriptions, searching knowledge articles, checking customer history, and deciding whether to escalate. The business problem is not "we need a chatbot." The problem is slow triage, inconsistent prioritization, and long resolution times.

Example 2: Finance operations. A fund operations team manually reviews reconciliation breaks between internal ledgers and custodian reports. Analysts spend time identifying break categories, checking previous notes, and preparing status updates. The business problem is delayed break resolution and inconsistent documentation, especially near month-end close.

Checklist questions:

- Is the problem clearly tied to a business outcome?
- Can stakeholders explain why existing tools or automation are not enough?
- Is there sufficient volume or value to justify an agentic solution?

- Are there clear pain points that an agent can improve through reasoning, tool use, or workflow coordination?
- Have risks been identified if the agent makes an incorrect recommendation or action?

2.2 Define the Agent's Goal and Scope

Once the business problem is clear, define what the agent is expected to achieve. The goal should describe the outcome, not only the activity. For example, “classify support tickets” is an activity, while “reduce time to first response by automatically triaging tickets and drafting recommended replies for human approval” is an outcome-oriented goal.

Scope is equally important. A useful agent has a clear operating boundary. It should be obvious what the agent can do, what it cannot do, when it must ask for help, and when it must escalate to a human. Without scope boundaries, agentic systems can become unpredictable, difficult to test, and hard to govern.

- **Primary goal:** What business outcome should the agent achieve?
- **Allowed tasks:** What actions may the agent perform independently?
- **Restricted tasks:** What actions are outside the agent's authority?
- **Autonomy level:** Will the agent only recommend, draft, execute with approval, or execute automatically within limits?
- **Escalation triggers:** When should the agent stop and involve a human?

- **Time horizon:** Is the agent completing a single task, managing a workflow over days, or continuously monitoring an environment?

Example scope definition: A procurement assistant agent may read purchase requests, check supplier policy, compare request details against budget codes, ask clarifying questions, and recommend approval or rejection. However, it may not approve purchases above a defined threshold, add new suppliers, modify payment terms, or bypass compliance review.

Good agent goal statement: "The agent will reduce procurement review cycle time by pre-checking purchase requests against policy, identifying missing information, drafting clarification questions, and routing complete requests to the correct approver."

Poor agent goal statement: "The agent will handle procurement." This is too broad because it does not define tasks, authority, systems, users, controls, or success criteria.

Checklist questions:

- Is the agent's goal written as a measurable business outcome?
- Are the agent's responsibilities specific enough to test?
- Are prohibited actions explicitly documented?
- Is the autonomy level appropriate for the risk of the task?
- Are human approval points defined for sensitive, irreversible, or high-value actions?

- Are escalation rules defined for uncertainty, missing data, exceptions, or policy conflicts?

2.3 Identify Users, Systems, and Success Metrics

An agentic AI use case must identify who will use the agent, which systems it will interact with, and how success will be measured. This step prevents vague architecture decisions and helps teams design access controls, integration patterns, user experiences, and evaluation methods.

Users include not only the people who chat with or trigger the agent, but also reviewers, approvers, administrators, auditors, and downstream teams affected by the agent's actions. Different users may need different permissions, explanations, notifications, and override options.

- **Primary users:** People who directly interact with the agent, such as support agents, analysts, HR advisors, or customers.
- **Business owners:** Leaders accountable for the process outcome and agent behavior.
- **Human reviewers:** Users who approve, reject, or modify agent recommendations.
- **Administrators:** Teams that manage configuration, prompts, tools, access, and monitoring.

- **Auditors and compliance teams:** Stakeholders who need traceability, evidence, and policy alignment.

Systems include the applications, databases, knowledge sources, APIs, workflow tools, and communication channels the agent needs to read from or write to. Each integration should be classified by access type: read-only, write with approval, or write automatically within defined limits.

- **Knowledge sources:** Policies, SOPs, product documentation, FAQs, training materials, and historical cases.
- **Operational systems:** CRM, ERP, ticketing systems, HR systems, finance platforms, data warehouses, and workflow tools.
- **Communication channels:** Email, Teams, Slack, web chat, customer portals, or service desks.
- **Monitoring systems:** Logs, dashboards, observability tools, incident management tools, and audit repositories.
- **Security systems:** Identity providers, access management tools, data loss prevention controls, and approval workflows.

Success metrics should combine business value, operational performance, quality, adoption, and risk. Metrics should be defined before the pilot so that teams can evaluate whether the agent is improving the process or only creating a new layer of complexity.

- **Business value:** Cost reduction, revenue protection, faster cycle time, fewer manual hours, or improved service levels.
- **Quality:** Accuracy of classification, correctness of recommendations, fewer rework cases, and better documentation completeness.
- **User experience:** User satisfaction, adoption rate, response usefulness, and reduction in repetitive work.
- **Risk and control:** Number of escalations handled correctly, policy violations avoided, audit trail completeness, and approval compliance.
- **Technical performance:** Latency, tool success rate, retrieval quality, failure rate, retry rate, and cost per task.

Example metrics for a reconciliation agent:

- Reduce average break investigation time from 30 minutes to 12 minutes.
- Classify at least 90% of standard break types correctly during pilot testing.
- Generate complete investigation notes for at least 95% of reviewed breaks.
- Escalate all high-value or policy-sensitive breaks to a human reviewer.
- Maintain a searchable audit trail for every recommendation, source document, and action taken.

Checklist questions:

- Have all user groups and affected stakeholders been identified?
- Are system integrations listed with read/write access requirements?
- Are sensitive systems protected with role-based permissions and approval controls?
- Are business, quality, user experience, risk, and technical metrics defined?
- Is there a baseline measurement from the current process for comparison?
- Are success metrics realistic, measurable, and linked to the agent's stated goal?

3. Map Your Agentic AI Architecture

After the use case is defined, the next step is to map the architecture. A practical agentic AI architecture is usually organized into layers so that teams can separate user experience, orchestration, agent behavior, memory, tools, data, models, governance, and monitoring. This layered view helps stakeholders understand where decisions happen, where data flows, where risks exist, and which components must be tested before deployment.

A useful architecture map should show more than technical components. It should show the path from user intent to agent reasoning, context retrieval, tool execution, human approval, system update, monitoring, and audit evidence. This makes it easier to identify where the agent may fail, where controls are needed, and where business ownership must be assigned.

Architecture Layer	Primary Question	Example Design Decision
Application	How will users interact with the agent?	Use a service portal for request intake and an approval screen for high-risk actions.
Orchestration	How will work move across steps?	Use a workflow engine to route requests, track state,

		trigger approvals, and log outcomes.
Agent	What role does each agent perform?	Separate planning, policy review, and execution into distinct agent responsibilities.
Context and Memory	What information does the agent use and retain?	Use retrieval for policy documents and limit persistent memory to approved case metadata.
Tools and Actions	What can the agent do in external systems?	Allow read access broadly but require approval before updating financial or customer records.
Data	Which data sources are trusted?	Prioritize governed systems of record over informal documents or outdated spreadsheets.

Model	Which model capability is needed?	Use a stronger reasoning model for policy interpretation and a lower-cost model for classification.

3.1 Application Layer

The application layer is the user-facing part of the agentic AI system. It defines how users interact with the agent, where the agent appears, what information is shown, and how users approve, reject, or refine agent outputs. This layer may include a chat interface, embedded assistant, workflow screen, email experience, service portal, mobile app, or internal operations dashboard.

- **User entry points:** Define whether users access the agent through chat, workflow forms, ticketing systems, portals, or embedded application panels.
- **Interaction design:** Decide whether the agent asks questions, recommends actions, presents options, or executes tasks in the background.
- **Approval experience:** Provide clear buttons, review screens, or workflow steps for users to approve sensitive actions.
- **Explanation display:** Show why the agent made a recommendation, including sources, assumptions, confidence level, and next steps.
- **Error handling:** Give users clear messages when the agent cannot complete a task or needs human assistance.

Example: In a claims processing use case, the application layer may show the claim summary, missing documents, recommended decision, policy references, and an approval button for the claims analyst. The user should not need to inspect backend logs to understand what the agent is proposing.

Detailed example: For an HR policy assistant, the application layer may include an employee self-service chat interface, an HR advisor review panel, and a case management screen. Employees can ask questions about leave, benefits, or policy interpretation. The agent should answer with plain-language guidance, show the source policy used, and route complex or sensitive cases to HR rather than making a final decision.

Application layer checklist:

- Is the agent embedded where users already work?
- Does the interface clearly distinguish between recommendations, drafts, and completed actions?
- Are source references, assumptions, and confidence indicators visible to reviewers?
- Can users correct the agent, ask follow-up questions, or escalate to a human?
- Are error messages written in business language rather than technical language?

3.2 Orchestration Layer

The orchestration layer coordinates the agentic workflow from start to finish. It decides which steps happen, which agent or model is called, when context is retrieved, which tool is used, how errors are handled, and when human approval is required. In enterprise systems, orchestration is often the control plane that makes agent behavior reliable, observable, and governable.

- **Workflow routing:** Direct tasks to the right agent, process, or human queue.
- **State management:** Track the current step, previous decisions, retrieved context, tool results, and pending actions.
- **Retries and fallbacks:** Define what happens when a model response fails, a tool times out, or a system is unavailable.
- **Human checkpoint routing:** Pause workflows for approval when risk, value, uncertainty, or policy rules require review.
- **Observability:** Record traces, step outputs, tool calls, decision paths, and timing information.

Example: A finance reconciliation agent may follow an orchestration path such as: receive break data, retrieve prior notes, classify break type, check tolerance policy, draft investigation summary, recommend action, route high-value breaks for review, and update the workflow status after approval.

Detailed example: In a fund operations reconciliation workflow, orchestration may begin when a new break file is uploaded. The orchestrator assigns the case to a classification step, retrieves prior break history, checks materiality thresholds, asks the agent to draft an investigation note, routes high-value items for approval, and then updates the break tracker only after reviewer confirmation.

Orchestration layer checklist:

- Is every workflow step documented from trigger to final outcome?
- Does the system track state across multi-step and multi-day workflows?
- Are retries, timeouts, and fallback routes defined for each tool or system dependency?
- Are approval gates triggered by risk, value, confidence, or policy rules?
- Can support teams inspect the workflow trace when users report an issue?

3.3 Agent Layer

The agent layer defines the agent roles, responsibilities, instructions, reasoning patterns, and operating boundaries. A simple system may have one agent, while a more complex system may use multiple specialized agents, such as a planner agent, research agent, compliance reviewer, tool execution agent, and summarization agent.

- **Role definition:** Describe the job of each agent in plain language.

- **System instructions:** Define tone, behavior, constraints, escalation rules, and source-use expectations.
- **Decision authority:** Clarify whether the agent recommends, drafts, validates, or executes.
- **Specialization:** Separate high-risk responsibilities so that one agent does not perform every task without review.
- **Coordination:** Define how agents hand off work, share context, and avoid conflicting actions.

Detailed example: A contract review solution may use a drafting agent to summarize clauses, a risk agent to identify non-standard terms, a compliance agent to compare clauses against policy, and a routing agent to decide whether legal review is required. This division of responsibility reduces the risk that one broad agent will summarize, judge, and approve its own output without independent review.

Agent layer checklist:

- Does each agent have a clearly defined role and boundary?
- Are agent instructions version-controlled and reviewed before release?
- Are high-risk decisions separated from execution actions?
- Does the architecture prevent agents from overriding policy or reviewer decisions?

- Are handoffs between agents logged with context and rationale?

3.4 Context & Memory Layer

The context and memory layer determines what the agent knows during a task and what it can remember across tasks. Context may include the user's request, conversation history, retrieved documents, system outputs, tool results, and business rules. Memory may be short-term for the current session or long-term for persistent preferences, case history, or process learning.

- **Working memory:** Holds temporary information needed to complete the current task.
- **Session memory:** Maintains context across a conversation or workflow instance.
- **Long-term memory:** Stores durable knowledge such as user preferences, case patterns, or approved summaries.
- **Retrieval context:** Supplies relevant content from documents, databases, knowledge bases, or vector indexes.
- **Memory controls:** Define what may be stored, for how long, who can access it, and how it can be deleted.

Checklist questions: Is the agent using current and trusted context? Is sensitive information excluded from long-term memory unless approved? Can users or administrators inspect, correct, or remove stored memory when required?

Detailed example: A customer success agent may need short-term memory of the current conversation, retrieval access to product documentation, and account-specific context from the CRM. However, it should not permanently store sensitive customer complaints or confidential commercial terms unless there is a governed business reason and a defined retention policy.

Context and memory checklist:

- Are trusted knowledge sources clearly identified and prioritized?
- Is memory separated into temporary session state and approved long-term memory?
- Are memory retention, deletion, and correction rules documented?
- Does the agent avoid using outdated policies or stale case history?
- Can the organization explain which context influenced a decision?

3.5 Tool & Action Layer

The tool and action layer contains the capabilities the agent can use to interact with the outside world. Tools may include APIs, search connectors, databases, calculators, workflow engines, email systems, ticketing platforms, robotic process automation bots, or custom business services. This layer is where an agent moves from "answering" to "doing."

- **Tool catalog:** Maintain an approved registry of tools the agent can call.

- **Input validation:** Check tool inputs before execution to prevent malformed or unsafe actions.
- **Permission boundaries:** Limit each tool based on role, task, environment, and approval status.
- **Execution controls:** Use sandboxing, rate limits, transaction thresholds, and confirmation steps for sensitive actions.
- **Rollback and correction:** Define how incorrect actions can be reversed or remediated.

Detailed example: A service desk agent may use tools to search a knowledge base, check device inventory, reset a password, create a ticket, and notify a user. The architecture should treat each tool differently. Searching knowledge articles may be low risk, but password resets and account changes require stronger identity verification, approval rules, and complete logging.

Tool and action checklist:

- Is every tool listed in an approved tool registry?
- Are tool inputs validated before execution?
- Are write actions protected with confirmations, approvals, or transaction thresholds?
- Are tool failures visible to the user and support team?

- Are rollback, correction, or remediation steps defined for each action?

3.6 Data Layer

The data layer includes the structured, semi-structured, and unstructured information the agent uses. This may include enterprise documents, customer records, transaction data, policy manuals, tickets, emails, meeting notes, product data, and system logs. The data layer must be governed because an agent can only be as reliable as the information it can access.

- **Source inventory:** List all data sources and classify them by sensitivity, owner, freshness, and quality.
- **Access control:** Enforce least-privilege access and ensure the agent cannot retrieve data the user is not allowed to see.
- **Data quality:** Validate completeness, accuracy, timeliness, and consistency of key sources.
- **Retention rules:** Define how long retrieved context, logs, and generated outputs are retained.
- **Lineage:** Track which source records influenced each recommendation or action.

Detailed example: In a finance close assistant, the data layer may include the general ledger, reconciliation files, close calendar, policy documents, prior period variance explanations, and approval logs. The agent should distinguish system-of-record data

from working files and should not rely on unsupported spreadsheets when governed data is available.

Data layer checklist:

- Are data owners assigned for every source used by the agent?
- Are sources classified by sensitivity, reliability, and freshness?
- Does the agent respect user-level permissions when retrieving data?
- Are data quality checks performed before high-impact recommendations?
- Is source lineage retained for audit and troubleshooting?

3.7 Model Layer

The model layer includes the foundation models, embedding models, classification models, rerankers, domain-specific models, and any model-routing logic used by the agent. Model choice should be based on task requirements, risk level, latency, cost, accuracy, privacy, and deployment constraints.

- **Model fit:** Match the model to the task, such as reasoning, summarization, classification, extraction, or code generation.
- **Model routing:** Use different models for different steps when cost, latency, or accuracy requirements vary.
- **Evaluation:** Test model outputs against benchmark cases, policy expectations, and human-reviewed examples.

- **Privacy and hosting:** Decide whether the model runs through a managed service, private deployment, or on-premises environment.
- **Fallback strategy:** Define what happens if the selected model is unavailable or produces low-confidence output.

Detailed example: A triage agent may use a fast, lower-cost model to classify routine requests, a stronger reasoning model for complex policy interpretation, and an embedding model for document retrieval. This avoids using the most expensive model for every step while still reserving higher reasoning capability for decisions that require it.

Model layer checklist:

- Is the selected model appropriate for the task complexity and risk?
- Are model outputs evaluated against representative test cases?
- Are fallback models or fallback workflows defined?
- Are latency, cost, privacy, and hosting requirements documented?
- Are model changes tested before production rollout?

4. Evaluate Core Architecture Components

Once the architecture layers are mapped, evaluate the core components that determine whether the agent will behave reliably in real workflows. These components include planning, memory, integrations, feedback, and human checkpoints. Weakness in any one of these areas can make an agent difficult to trust or scale.

Evaluation should focus on the behavior of the entire agentic system, not only the quality of the final answer. A successful agent must choose the right steps, retrieve the right context, use the right tools, handle uncertainty, explain its reasoning, and stop when human judgment is required. This is especially important because agent failures often occur in the middle of a workflow rather than in the final response.

Component	What to Evaluate	Example Evidence
Planning and Reasoning	Whether the agent decomposes tasks correctly and adapts to new information.	Workflow traces, planning steps, reviewer comments, and test case outcomes.
Memory Management	Whether memory is relevant, current, secure, and correctable.	Memory records, retention settings, correction logs, and privacy review evidence.

Tool Registry and Integrations	Whether tools are approved, reliable, permissioned, and logged.	Tool registry, API logs, access reviews, and error reports.
Feedback Loops	Whether user, reviewer, and system feedback improve quality safely.	Feedback dashboards, issue trends, evaluation datasets, and improvement backlog.
Human-in-the-Loop Checkpoints	Whether human review occurs at the right moments and is properly recorded.	Approval logs, override reasons, escalation records, and audit trail samples.

4.1 Planning & Reasoning

Planning and reasoning define how the agent decomposes a goal into steps, evaluates options, chooses actions, and adapts when information changes. For low-risk tasks, simple step-by-step prompting may be enough. For high-risk or complex workflows, the agent may need explicit planning logic, state tracking, validation rules, and human review.

- Can the agent break down a complex request into clear steps?
- Does the agent explain assumptions before taking action?
- Does the agent check whether required information is missing?
- Can the agent revise its plan if a tool result contradicts the initial assumption?

- Are high-risk reasoning steps reviewed or constrained by rules?

Detailed example: In a procurement use case, the agent should not simply recommend approval because a request looks complete. It should reason through budget availability, supplier status, purchase category, policy thresholds, missing documentation, and approval authority. If the supplier is new or the amount exceeds a threshold, the plan should route the case to procurement or finance review.

Planning and reasoning checklist:

- Does the agent create a plan before taking multi-step action?
- Does the agent identify missing information before proceeding?
- Does it update the plan when tool results conflict with assumptions?
- Does it explain the rationale in a way a reviewer can understand?
- Are reasoning failures captured and used for future evaluation cases?

4.2 Memory Management

Memory management determines how the agent stores, retrieves, updates, and forgets information. Poor memory design can lead to outdated context, privacy risk, inconsistent behavior, or incorrect personalization. Good memory design makes the agent more useful while keeping control over what is retained.

- **Store only what is necessary:** Avoid retaining sensitive or irrelevant information.

- **Separate memory types:** Keep task state, user preferences, case notes, and organizational knowledge distinct.
- **Use freshness rules:** Prevent the agent from relying on outdated facts or expired policies.
- **Support correction:** Allow authorized users to update or delete incorrect memory.
- **Audit memory use:** Record when memory influenced a recommendation or action.

Detailed example: A sales assistant may remember that a user prefers short account summaries, but it should not remember confidential negotiation details unless those details are stored in an approved CRM record. Memory should support productivity without becoming an uncontrolled shadow database.

Memory management checklist:

- Is each memory type linked to a clear business purpose?
- Are sensitive memory fields masked, restricted, or excluded?
- Are freshness and expiry rules applied to retained memory?
- Can users or administrators correct inaccurate memory?
- Is memory use visible in audit logs when it affects a recommendation?

4.3 Tool Registry & Integrations

A tool registry is an approved inventory of systems, APIs, functions, and services that an agent can use. It should describe the purpose of each tool, required inputs, output format, permissions, risk rating, owner, and logging requirements. Without a registry, tool use can become uncontrolled and difficult to audit.

- Is every tool approved by a business and technical owner?
- Are write actions separated from read-only actions?
- Are tool inputs validated before execution?
- Are tool outputs checked before they are used in decisions?
- Are tool calls logged with user, agent, time, input, output, and result status?

Detailed example: A customer operations agent may have access to tools for customer lookup, order status, refund eligibility, email drafting, and refund submission. The tool registry should mark customer lookup as read-only, email drafting as draft-only, and refund submission as write action requiring approval above a defined value.

Tool registry checklist:

- Does the registry list owner, purpose, risk rating, inputs, outputs, and permissions?
- Are tools grouped by read, draft, write-with-approval, and write-automatic categories?

- Are sensitive tools protected by additional authentication or approval?
- Are failed tool calls reviewed for root cause and user impact?
- Are unused or risky tools periodically removed from the approved catalog?

4.4 Feedback Loops

Feedback loops help the agent improve and help the organization detect issues. Feedback can come from users, reviewers, system outcomes, quality checks, audit findings, incident reports, or automated evaluations. The goal is not to let the agent learn uncontrolled from every interaction, but to capture evidence that supports safe improvement.

- **User feedback:** Thumbs-up, thumbs-down, comments, corrections, and usability ratings.
- **Reviewer feedback:** Reasons for approving, editing, rejecting, or escalating agent recommendations.
- **Outcome feedback:** Whether the final business result matched the agent's expectation.
- **Evaluation feedback:** Test results from benchmark datasets, scenario tests, or regression checks.
- **Governance feedback:** Findings from audits, risk reviews, security assessments, or incident investigations.

Detailed example: A claims agent may receive feedback when reviewers edit recommendations, reject a decision, or escalate a case. Instead of treating feedback as free-form comments only, the system should capture structured reasons such as missing evidence, incorrect policy interpretation, poor tone, incomplete calculation, or unnecessary escalation.

Feedback loop checklist:

- Are feedback reasons structured enough to support analysis?
- Are high-severity failures reviewed by accountable owners?
- Are recurring feedback themes converted into backlog items?
- Are evaluation datasets updated with real failure examples?
- Is feedback used safely rather than allowing uncontrolled self-learning?

4.5 Human-in-the-Loop Checkpoints

Human-in-the-loop checkpoints define when a person must review, approve, or override the agent. These checkpoints are especially important when decisions affect customers, employees, finances, compliance obligations, safety, or legal rights. Human oversight should be purposeful and built into the workflow, not treated as an informal backup.

- Require approval for high-value, irreversible, regulated, or customer-impacting actions.

- Escalate when confidence is low, data is missing, policies conflict, or the agent detects ambiguity.
- Provide reviewers with the agent's reasoning, source evidence, and recommended next action.
- Allow humans to override agent decisions and record the reason for the override.
- Use reviewer decisions to improve prompts, retrieval, tools, controls, and training materials.

Detailed example: In a finance workflow, the agent may draft journal entry support, classify variance explanations, or recommend a reconciliation action. However, it should require human approval before posting entries, changing close status, or clearing material reconciliation breaks. The approval record should include the reviewer, date, evidence reviewed, and reason for approval or override.

Human-in-the-loop checklist:

- Are approval checkpoints tied to risk, value, confidence, or regulatory requirements?
- Do reviewers receive enough context to make a decision efficiently?
- Can reviewers approve, reject, edit, escalate, or request more information?
- Are override reasons captured in a structured format?

- Are approval logs available for audit, quality review, and incident investigation?

5. Governance & Security

Governance and security must be designed into agentic AI systems from the start. Agents can access data, make recommendations, call tools, and trigger actions across enterprise systems. That means organizations need clear ownership, access controls, policy guardrails, oversight routines, compliance alignment, and audit evidence before agents are scaled.

A strong governance model treats an AI agent as an operational actor with delegated authority, not just as a software feature. This means the organization should know who owns the agent, what the agent is allowed to do, what data it can access, how its actions are monitored, and how incidents are handled. Governance should also scale with risk: a low-risk drafting assistant does not need the same controls as an agent that can update financial records or trigger customer-impacting actions.

Governance Area	Key Decision	Practical Example
Ownership	Who is accountable for the agent?	Assign a business owner for outcomes, a technical owner for reliability, and a risk owner for controls.
Risk Tiering	How risky is the use case?	Classify internal drafting as low risk, customer-impacting workflows as elevated risk, and

		regulated financial decisions as high risk.
Access Control	What can the agent read or write?	Allow read access to policy documents but require approval before updating customer, HR, or finance systems.
Monitoring	How will behavior be observed?	Track tool calls, policy violations, failed actions, escalation rates, and unusual activity.
Incident Response	What happens when something goes wrong?	Define suspension, rollback, evidence preservation, communication, and remediation steps.

5.1 Tool Access & Permissions

Tool access should follow least-privilege principles. The agent should only access the tools and data needed for its approved purpose, and write actions should be more tightly controlled than read actions. Permissions should be tied to the user, agent role, task type, environment, and approval status.

- Define read-only, write-with-approval, and write-automatic permission levels.
- Use role-based or attribute-based access control for users and agents.
- Prevent agents from inheriting excessive administrator privileges.
- Review permissions regularly and remove unused access.
- Separate development, testing, and production tool access.

Detailed example: A customer refund agent may need access to order history, refund policy, customer profile, and payment status. However, it should not automatically issue refunds above a defined value without approval. A practical permission model might allow the agent to read order data, draft a refund recommendation, submit low-value refunds automatically within policy, and route high-value or disputed refunds to a supervisor.

Tool access and permissions checklist:

- Is every agent assigned a distinct identity rather than sharing generic service credentials?
- Are permissions based on the user's role, the agent's approved purpose, and the sensitivity of the action?
- Are high-risk tools protected by approval gates, transaction limits, and additional verification?
- Are development, testing, and production credentials separated?

- Can access be revoked immediately if the agent behaves unexpectedly?

5.2 Guardrails & Policy Controls

Guardrails are controls that prevent unsafe, non-compliant, or out-of-scope behavior. They can be implemented through prompts, rules, retrieval filters, tool permissions, content filters, approval workflows, policy engines, and monitoring alerts. Guardrails should be specific to the agent's risk profile and business context.

- **Scope guardrails:** Stop the agent from performing tasks outside its approved purpose.
- **Data guardrails:** Prevent exposure of restricted, confidential, or unnecessary data.
- **Action guardrails:** Require validation or approval before sensitive tool execution.
- **Policy guardrails:** Enforce business rules, compliance thresholds, and escalation paths.
- **Output guardrails:** Reduce unsupported claims, hallucinations, policy violations, or inappropriate responses.

Detailed example: A policy interpretation agent should be prevented from answering questions outside approved policy sources. If a user asks the agent to ignore company policy, reveal restricted information, or perform an action outside its scope, the

guardrail should stop the request, explain the limitation, and route the user to the correct process if appropriate.

Guardrail Type	Purpose	Example Control
Input Guardrail	Screen user requests before processing.	Detect prompt injection, requests for restricted data, or instructions to bypass policy.
Retrieval Guardrail	Control what knowledge the agent can use.	Retrieve only from approved repositories and filter by user permissions.
Reasoning Guardrail	Constrain decisions during task execution.	Require policy checks before recommendations involving finance, HR, legal, or customer impact.
Action Guardrail	Control tool execution.	Block write actions unless required fields, approvals, and thresholds are satisfied.

Output Guardrail	Review final responses or actions.	Check for unsupported claims, sensitive data leakage, and missing disclaimers.

5.3 Human Oversight

Human oversight ensures that accountable people remain responsible for important decisions and that the agent does not operate beyond approved boundaries. Oversight should include named owners, escalation paths, review routines, performance reviews, and incident response procedures.

- Assign an executive owner, product owner, technical owner, and risk owner.
- Define who approves agent changes, tools, prompts, data sources, and production releases.
- Require reviewer sign-off for sensitive workflows.
- Track override rates, escalation rates, and unresolved exceptions.
- Schedule periodic governance reviews after deployment.

Detailed example: In a procurement workflow, the agent may recommend whether a purchase request is complete, but a procurement manager should approve supplier onboarding, exceptions to policy, or purchases above threshold. Human oversight

should not be limited to occasional review; it should be built into the workflow where judgment, accountability, and business context matter.

- Define which decisions require mandatory human review.
- Give reviewers access to evidence, sources, confidence indicators, and recommended actions.
- Capture approval, rejection, edit, escalation, and override reasons.
- Review human override trends to identify weak prompts, poor retrieval, unclear policy, or inadequate training.
- Ensure business owners regularly review whether the agent remains aligned with the intended operating model.

5.4 Security, Privacy & Compliance

Security, privacy, and compliance controls protect the organization, users, customers, and regulated data. Agentic systems should align with existing security architecture, identity management, data governance, compliance frameworks, and incident response procedures. Controls should cover both the agent itself and the systems it can access.

- **Identity and access:** Authenticate users, agents, services, and tool calls.
- **Data protection:** Classify data, mask sensitive fields, encrypt where appropriate, and prevent unauthorized disclosure.
- **Privacy:** Limit collection, retention, and reuse of personal information.

- **Compliance:** Map requirements to applicable regulations, internal policies, audit standards, and contractual obligations.
- **Security monitoring:** Detect abnormal tool use, prompt injection attempts, data leakage, privilege misuse, and unexpected agent behavior.

Detailed example: An agent connected to email, document repositories, and customer systems may be exposed to indirect prompt injection through malicious content embedded in documents or messages. Security controls should prevent the agent from treating untrusted content as instructions, restrict what tools can be called from retrieved content, and alert security teams when the agent attempts unusual actions.

Security, privacy, and compliance checklist:

- Are agent identities, user identities, and service identities separately managed?
- Are sensitive data fields masked or excluded from prompts, logs, and long-term memory where appropriate?
- Are prompt injection, data exfiltration, tool misuse, and privilege escalation risks tested?
- Are compliance obligations mapped to controls, evidence, and responsible owners?
- Are logs retained in a secure, searchable, and tamper-resistant manner?

6. Testing & Deployment Readiness

Testing agentic AI systems requires more than checking whether the agent can answer a prompt. Teams must test workflows, tool calls, data retrieval, permissions, failure handling, human approvals, monitoring, and business outcomes. Deployment readiness means the agent is technically functional, operationally supportable, governed, secure, and measurable.

Deployment readiness should be treated as a release gate. Before an agent goes live, the team should prove that the agent can complete expected workflows, fail safely, respect permissions, generate usable logs, support rollback, and operate within cost and performance limits. Production readiness should include business sign-off, technical sign-off, security review, compliance review, and support readiness.

Testing Area	Purpose	Example Test
Workflow Testing	Confirm the agent completes real business journeys.	Run end-to-end cases from intake to recommendation, approval, and system update.
Adversarial Testing	Check resistance to unsafe or manipulative inputs.	Test prompt injection, policy bypass attempts, and

		malicious content in retrieved documents.
Failure Testing	Verify safe behavior when dependencies fail.	Simulate API outage, missing data, permission denial, and timeout scenarios.
Performance Testing	Check speed, cost, and scalability.	Measure latency, tool success rate, token usage, and cost per completed task.
Operational Testing	Confirm support and incident readiness.	Test rollback, kill switch, alert routing, escalation paths, and evidence preservation.

6.1 Workflow Testing

Workflow testing verifies that the agent can complete end-to-end business scenarios under realistic conditions. Test cases should include common requests, edge cases, missing information, conflicting instructions, policy exceptions, and multi-step tool use.

- Test complete journeys from user request to final outcome.
- Use real-world examples, not only ideal demo cases.

- Validate whether the agent retrieves the right context and cites the right source internally.
- Check whether approval steps trigger at the correct time.
- Confirm that outputs are understandable to business users.

Detailed example: For a service desk agent, workflow testing should include password reset requests, access requests, device issue tickets, incomplete user requests, urgent escalations, and duplicate tickets. The goal is to confirm that the agent not only answers correctly but also asks clarifying questions, uses the correct tools, avoids unauthorized actions, and routes exceptions properly.

Workflow testing checklist:

- Are test cases based on real historical scenarios and expected future use cases?
- Do tests include common, edge, ambiguous, and policy-sensitive cases?
- Does the agent ask for missing information instead of guessing?
- Are tool calls executed in the correct sequence?
- Do reviewers agree that outputs are useful, complete, and explainable?

6.2 Failure & Recovery Testing

Failure and recovery testing verifies how the agent behaves when something goes wrong. This is essential because production environments include unavailable systems,

incomplete data, ambiguous requests, rate limits, permission denials, and unexpected tool responses.

- Simulate tool timeouts, API errors, and system outages.
- Test missing, stale, inconsistent, or restricted data.
- Verify that the agent does not fabricate results when context is unavailable.
- Confirm fallback paths, retries, escalation, and user notifications.
- Validate rollback or remediation processes for incorrect actions.

Detailed example: If a reconciliation agent cannot access the custodian file, it should not invent a break explanation. It should clearly state that the required source is unavailable, preserve the current workflow state, notify the appropriate team, and either retry according to policy or route the case to a human reviewer.

Failure and recovery checklist:

- Does the agent fail safely when systems are unavailable?
- Are retry limits, timeout rules, and escalation paths documented?
- Does the agent avoid unsupported claims when retrieval fails?
- Can incorrect actions be reversed, corrected, or escalated?
- Is there a tested kill switch or suspension process for serious incidents?

6.3 Monitoring & Audit Logs

Monitoring and audit logs provide visibility after deployment. They help teams understand what the agent did, why it did it, which sources and tools were used, who approved actions, where failures occurred, and whether the agent remains within policy boundaries.

- Log user requests, agent outputs, retrieved sources, tool calls, approvals, overrides, and errors.
- Monitor latency, failure rates, escalation rates, policy violations, and cost per task.
- Create alerts for unusual behavior, repeated failures, sensitive data exposure, or unauthorized tool attempts.
- Ensure audit logs are tamper-resistant and retained according to policy.
- Make logs usable for support, risk review, incident response, and continuous improvement.

Detailed example: A production monitoring dashboard for an agent should show request volume, successful completion rate, escalation rate, tool failure rate, policy violations, average latency, cost per task, and reviewer override rate. These metrics help technical teams detect system issues and help business owners understand whether the agent is delivering value.

Monitoring and audit log checklist:

- Are all agent actions traceable from user request to final outcome?
- Are tool calls, retrieved sources, approvals, overrides, and errors logged?
- Are alerts configured for unusual activity, repeated failures, policy violations, and cost spikes?
- Can logs support audit, troubleshooting, incident response, and model evaluation?
- Are dashboards understandable to both technical and business stakeholders?

6.4 Performance & Success Metrics

Performance and success metrics determine whether the agent is delivering value safely. Metrics should be tracked during pilot, rollout, and steady-state operations. Use both technical metrics and business metrics so the team can see whether the agent is accurate, useful, reliable, secure, and cost-effective.

- **Technical:** Latency, uptime, tool success rate, retrieval quality, error rate, and model cost.
- **Business:** Cycle time reduction, case throughput, cost savings, service-level improvement, and productivity gains.
- **Quality:** Accuracy, completeness, consistency, rework rate, and reviewer acceptance rate.

- **Risk:** Escalation correctness, policy violations, unauthorized access attempts, and audit evidence completeness.
- **Adoption:** Active users, repeat usage, user satisfaction, training completion, and feedback trends.

Detailed example: A claims processing agent may be considered successful only if it reduces review time, maintains recommendation accuracy, improves documentation completeness, avoids policy breaches, and receives positive reviewer feedback. A narrow metric such as "number of responses generated" is not enough because it does not measure business value or control quality.

Metric Category	Example Metric	Why It Matters
Business Value	Average cycle time reduction	Shows whether the agent improves operational efficiency.
Quality	Reviewer acceptance rate	Indicates whether recommendations are useful and accurate.
Risk	Policy violation rate	Shows whether controls are preventing unsafe behavior.

Reliability	Tool failure rate	Highlights integration or dependency problems.
Adoption	Repeat usage rate	Shows whether users trust and continue using the agent.

7. Agentic AI Readiness Scorecard

The readiness scorecard converts the checklist into a practical assessment. Each category can be rated as **Ready**, **Partially Ready**, or **Not Ready**. The purpose is not to create a perfect score, but to expose gaps that must be addressed before scaling the agent.

The scorecard should be used during design review, pilot exit review, and production readiness review. It gives stakeholders a common language to discuss whether the agent is ready to scale. The scorecard should be evidence-based: teams should not mark an item as Ready unless they can point to documents, test results, logs, approvals, dashboards, or other artifacts that prove readiness.

Score	Status	Meaning	Required Action
2	Ready	The requirement is implemented, tested, documented, and owned.	Proceed, monitor, and review periodically.
1	Partially Ready	The requirement is understood but incomplete, untested,	Create a remediation action with owner and target date.

		or weakly documented.	
0	Not Ready	The requirement is missing, unclear, or unsupported by evidence.	Resolve before pilot expansion or production deployment.

7.1 Architecture Readiness

- The use case, users, systems, and success metrics are clearly defined.
- The application, orchestration, agent, memory, tool, data, and model layers are documented.
- Tool access and system integrations are mapped with read/write permissions.
- Planning, memory, feedback, and human checkpoint logic are tested.
- Architecture decisions are traceable to business goals and risk requirements.

Evidence examples: architecture diagram, use case definition, system integration map, agent role definitions, prompt/version history, tool registry, model selection rationale, and workflow trace samples.

7.2 Governance Readiness

- Ownership, accountability, and decision rights are assigned.
- Guardrails, policies, permissions, and approval workflows are documented.

- Privacy, security, compliance, and data retention requirements are addressed.
- Audit logs capture agent reasoning, source use, tool calls, approvals, and overrides.
- Incident response and escalation procedures are defined.

Evidence examples: ownership matrix, risk tiering assessment, access review records, guardrail configuration, approval workflow design, privacy review, security review, compliance mapping, and incident response playbook.

7.3 Deployment Readiness

- Workflow testing, failure testing, and user acceptance testing are complete.
- Monitoring dashboards, alerts, and operational support procedures are in place.
- Performance, quality, risk, cost, and adoption metrics are baselined.
- Rollback, remediation, and change management procedures are ready.
- Users, reviewers, support teams, and administrators are trained.

Evidence examples: test results, user acceptance feedback, monitoring dashboards, alert routing rules, rollback plan, kill switch test, support runbook, training materials, and pilot exit report.

7.4 Overall Readiness Assessment

Use the overall readiness assessment to decide whether the agent should proceed to pilot, limited release, full production, or redesign. A simple scoring approach is to assign

2 points for Ready, 1 point for Partially Ready, and 0 points for Not Ready across each checklist item.

A useful readiness assessment should combine the numeric score with qualitative judgment. For example, an agent may score highly overall but still be blocked from production if one high-risk control is missing, such as approval for financial postings or controls preventing restricted data exposure. Critical control failures should be treated as deployment blockers even if the average score looks acceptable.

- **80–100% readiness:** Proceed to controlled deployment with active monitoring.
- **60–79% readiness:** Proceed only with a limited pilot and a clear remediation plan.
- **40–59% readiness:** Resolve major architecture, governance, or testing gaps before deployment.
- **Below 40% readiness:** Redesign the use case, architecture, or governance model before continuing.

Recommended scorecard practice: Review the scorecard with business, technology, security, compliance, and operations stakeholders. Capture owners and due dates for every Partially Ready or Not Ready item. Reassess after remediation and before any expansion in users, tools, data access, or autonomy level.

Conclusion

Key Takeaways

- Agentic AI architecture should begin with a clearly defined business problem, not a technology preference.
- Reliable agents require layered architecture across application, orchestration, agents, memory, tools, data, and models.
- Governance, security, privacy, and human oversight must be embedded by design.
- Testing must include workflow behavior, failure handling, permissions, monitoring, and auditability.
- A readiness scorecard helps teams decide whether to pilot, deploy, or redesign before scaling.

Recommended Next Steps

- Select one high-value use case and complete the use case definition section.
- Create a layered architecture diagram showing users, agents, tools, data, models, and controls.
- Build a tool registry and classify each integration by risk and permission level.
- Define human-in-the-loop checkpoints for high-risk actions.

- Run workflow, failure, governance, and security readiness reviews before production deployment.

Build Reliable Agentic AI Systems with Confidence

Agentic AI can create significant value when it is designed with clear goals, reliable architecture, trusted data, secure tool access, measurable outcomes, and accountable oversight. The goal is not simply to build agents that act autonomously. The goal is to build agents that act appropriately, transparently, securely, and in alignment with business objectives.

GSDC
Global Skill Development Council

AGENTIC AI PROFESSIONAL CERTIFICATION

AGENTIC AI IS BASED ON THE IDEA OF CREATING AI THAT CAN THINK AND ACT ON ITS OWN TO GET THINGS DONE, LIKE A HELPFUL ASSISTANT.

ABOUT GSDC CERTIFICATION

LIFETIME VALIDITY

GSDC Certification is an globally accreditted certification with lifetime validity.

EBOOK

Extensive and exclusive Ebook created by world's experts to help you with understanding core concepts.

CREATED BY EXPERTS

GSDC certifications are created and authored by world's leading experts in the field.

LEARNING MATERIALS

Get access to learning materials such as videos, ebooks, templates, and practice exams, which will help you clear the certification exam.

LEARNING OBJECTIVE

- Gain insights into autonomous decision-making processes
- Apply knowledge using ready-to-implement templates
- Demonstrate ability to work with Agentic AI models
- Validate your skills wit

Enroll now with the code **LEARN20** To avail **20%** discount

Enroll Now

www.gsdcouncil.org