Governing Enterprise AI Skills

Risk, Reuse, and Control in Agentic AI

Version 1.4 | May 2026

Prepared by Jan Uyttenhove / Insidin

Table of Contents

1. Abstract

2. Introduction: when productivity becomes a control problem

3. What is an AI skill?

4. Why skills are different from traditional software

5. The risk model: from useful reuse to unmanaged supply chain

- 5.1 Hidden instructions and prompt injection
- 5.2 Embedded secrets and confidential information
- 5.3 Unsafe code and dependency risk
- 5.4 The confused deputy problem and privilege misuse
- 5.5 Indirect action chains
- 5.6 Internal trust and uncontrolled spread

6. Why existing controls are not enough

7. Governance model: skills as governed enterprise artifacts

8. Reference architecture: controlled skill lifecycle and runtime enforcement

9. Practical adoption path

10. Conclusion

11. References

Abstract

Enterprises are beginning to use AI assistants and agentic systems not only for isolated conversations, but also as reusable work environments. Employees can create prompts, workflows, helper scripts, data-processing routines, and small pieces of automation, then share them with colleagues as "skills". This looks like a natural productivity pattern. It is also the beginning of a new internal supply chain.

A skill is not just documentation. It can contain instructions, code, examples, assumptions, access patterns, and operational knowledge. When such a skill is reused inside an AI assistant or agentic platform, it can influence behavior at runtime. That creates a different risk model from traditional software, because large language models do not enforce a clean boundary between instructions and data. User input, retrieved content, examples, tool results, and skill instructions can all become part of the model's operating context.

The enterprise risk is therefore not limited to malicious skills. The larger risk is that informal personal automation becomes shared enterprise behavior without ownership, review, versioning, policy enforcement, or auditability. Skills should be governed as first-class software-like artifacts, but software governance alone is not enough. Enterprises need lifecycle control, prompt and context review, secret detection, code scanning, controlled distribution, runtime policy enforcement, and audit logging.

Introduction: when productivity becomes a control problem

Employees will create AI skills because the value is obvious. A person discovers a better way to summarize a contract, transform a spreadsheet, query a dataset, generate test data, prepare a report, or interact with an internal system. The first version may be nothing more than a good prompt. The next version may include examples. A later version may include Python code, an API call, a file template, or a sequence of steps that an AI assistant can repeat.

At that point the skill no longer represents only a personal shortcut. It becomes a reusable way of working. When it is shared with colleagues, copied into a team space, added to a repository, or embedded into an AI agent, it becomes part of the enterprise operating environment. The organization may not call it software, but it starts behaving like a software artifact. It can be reused, modified, distributed, trusted, and executed.

The governance problem starts exactly there. Enterprises already know how unmanaged spreadsheets, macros, scripts, and shadow IT can become business-critical without control. AI skills accelerate the same pattern, but with a more complex risk profile. They can contain not only logic, but also natural-language instructions that influence model behavior. They can contain code, but also examples that reveal internal data. They can encode a process, but also assumptions about who is allowed to do what. They can be shared casually, while being executed by systems that have access to enterprise data, tools, and workflows.

The question is not whether employees should be allowed to create reusable AI capabilities. That will happen, and blocking it entirely would remove much of the value of enterprise AI. The question is whether the enterprise recognizes that personal automation becomes enterprise risk once it is shared, reused, and executed in a business context.

What is an AI skill?

An AI skill is a reusable capability that helps an AI assistant or agent perform a task. It can be very simple, such as a prompt that tells the assistant how to summarize meeting notes. It can also be more advanced, combining instructions, examples, scripts, configuration, API calls, or workflow steps.

A practical skill could say: "Take a vendor invoice, extract the supplier, amount, purchase order, VAT details, and payment deadline, then prepare a summary for review." A more advanced version could include a Python parser, a connection to a document repository, a validation rule against a finance system, and instructions for preparing a message to the responsible team.

This hybrid nature is the important part. A skill is not only a prompt. It is not only code. It is packaged know-how prepared for execution by an AI system. It may contain natural language, deterministic logic, business examples, tool instructions, data formats, internal terminology, and access assumptions in one artifact.

That makes skills powerful. It also makes them difficult to govern with existing controls. A software scanner may inspect the Python code, but ignore the prompt. A data loss prevention tool may detect some secrets, but miss sensitive business semantics hidden in examples. A human reviewer may understand the business purpose, but not recognize that a prompt instruction can manipulate tool use. A platform administrator may approve an AI capability once, while the real risk depends on who executes it, for what purpose, against which data, and through which tools.

Why skills are different from traditional software

Traditional software is built around a relatively clear separation between instructions and data. Developers write code. The system executes that code. User input is processed by the code, but the input does not normally rewrite the instruction set of the program. Security engineering has spent decades strengthening that boundary, because many classic vulnerabilities arise when input is treated as instruction.

AI systems weaken that boundary by design. A large language model receives a context window containing system instructions, developer instructions, user prompts, retrieved documents, tool outputs, conversation history, examples, and sometimes memory. The model uses that whole context to decide what to generate next. It does not have a built-in, reliable, security-grade distinction between trusted instruction and untrusted data.

This is why prompt injection matters. Prompt injection is not just a trick where someone writes "ignore previous instructions". It is the broader problem that content provided to the model can alter its behavior in unintended ways. OWASP defines prompt injection as a vulnerability where prompts alter an LLM's behavior or output in unintended ways, including cases where the injection is not visible to a human reader but is still parsed by the model [1].

Skills inherit this problem because skills are themselves part of the model's operating context. A shared skill may contain safe-looking instructions that change how the AI interprets data, calls tools, or handles restrictions. It may also include examples that become behavioral patterns. The enterprise cannot assume that because a skill is written in natural language, it is harmless. In an AI system, natural language can act as executable behavior.

Behavior is therefore no longer defined only before execution. It can be influenced during execution by the context the model receives. That is the core shift. It explains why ordinary software lifecycle controls are necessary, but not sufficient.

The risk model: from useful reuse to unmanaged supply chain

The most important enterprise risk is not that every skill will be malicious. Most will be created with good intent. The more realistic risk is that skills will be created quickly, shared informally, copied across teams, and gradually become part of daily work without review or ownership. In that environment, insecure behavior becomes normalized through reuse.

Hidden instructions and prompt injection

A shared skill can contain instructions that manipulate the AI system in ways the user does not see or understand. The instruction may be explicit, hidden in a comment, embedded in an example, or introduced indirectly through a document that the skill processes.

A reporting skill, for example, may instruct the assistant to "include all relevant records" even when the user only requested aggregated results. A more hostile version could tell the assistant to ignore restrictions, reveal hidden context, or call a tool in a way that bypasses intended boundaries. The risk increases when the skill is executed by an agent that can retrieve data, send messages, write files, or call internal APIs.

This is not only a technical vulnerability. It is a governance problem. The enterprise loses the ability to know which instruction actually shaped the action: the user's request, the platform policy, the skill, a retrieved document, or an injected payload.

Embedded secrets and confidential information

Skills can easily contain credentials or confidential information, especially when they evolve from prototypes. A person may include an API key in a helper script, an internal endpoint in an example, a real customer case in a prompt, or a schema name that exposes sensitive system design. The skill is then copied and reused by others who do not realize what it contains.

This risk is familiar from traditional software engineering, where secrets are accidentally committed to repositories. With skills, the detection problem is broader. The sensitive content may not only be in code. It may be in natural-language instructions, comments, examples, templates, or test data. The artifact may be shared in a chat message or document instead of a repository, making normal scanning even less likely.

The result is a leakage path that looks like collaboration. A colleague shares a useful skill. Another colleague pastes it into a different AI assistant. A third person adapts it for another team. At no point does the organization necessarily know that internal information has moved into a new context.

Unsafe code and dependency risk

Many skills will include deterministic programming components. Python scripts, shell commands, notebook fragments, macros, and API helpers will be attractive because they make the skill more reliable and repeatable. This brings traditional software risk into the AI skill lifecycle.

The code may contain command injection vulnerabilities, unsafe file handling, insecure deserialization, weak error handling, outdated dependencies, or excessive permissions. A dependency may be malicious or compromised. A script may work correctly in the author's environment, but become dangerous when executed by another user with different access rights or different data.

The enterprise should not treat these code fragments as harmless because they are small. Small scripts often run with large privileges. In an agentic setting, the script may also be invoked indirectly by a model, based on context that the developer did not anticipate.

The confused deputy problem and privilege misuse

The confused deputy problem is central to agentic AI. A deputy is a system that acts on behalf of someone else. In this context, the AI agent is the deputy. The problem occurs when the deputy has legitimate access, but is tricked into using that access for the wrong purpose.

A user may ask an AI assistant to retrieve all relevant files for a project. The assistant may have access through the user's authenticated session, an enterprise connector, or a service account. If a shared skill or injected instruction manipulates the assistant's behavior, the assistant may collect data that the user did not explicitly intend to request, or that should not be used for that purpose.

The important point is that the attacker does not need direct access to the protected system. The attacker only needs to influence the system that already has access. This is why the phrase "the agent has access" is not a sufficient security argument. The correct question is whether this specific execution is allowed for this user, this purpose, this data, this tool, and this action.

In AI environments, the deputy is particularly easy to confuse because it takes instructions from context. The agent may be trusted, but the content influencing it may not be trustworthy.

Indirect action chains

Skills can create multi-step behavior. A skill may retrieve data, summarize it, call another tool, prepare a message, create a ticket, update a record, or trigger an approval. The business value comes from chaining these steps. The risk comes from the same chain becoming difficult to observe and control.

A manipulated input may not directly cause damage in the first step. It may influence the second step, which changes the third step, which eventually triggers an action. Traditional logging may show only the final API call, not the prompt, skill instruction, retrieved context, intermediate reasoning, or policy decision that led to it.

This creates unobservable execution paths. For management, that is one of the most serious consequences. The enterprise may still have systems of record, but lose visibility into how AI-mediated work reached those systems.

Internal trust and uncontrolled spread

Internal sharing is often seen as safer than public sharing. That assumption is weak. Internal sharing can increase risk because trust lowers scrutiny. People are more likely to reuse a skill from a colleague than a random internet source. They are also more likely to modify it quickly and redistribute it.

This creates trust amplification, privilege amplification, and scale amplification. The skill comes from a trusted colleague, is executed inside an environment with access to real enterprise data, and spreads through team channels faster than formal controls can follow. Within weeks, several departments may use variants of a skill without knowing which version is current, who owns it, what it accesses, or whether it contains confidential content.

Public skills are a supply chain risk. Internal skills are a supply chain risk and an accountability risk. They turn informal workarounds into operational practice.

Why existing controls are not enough

Enterprises should reuse the discipline of software engineering. Ownership, versioning, repositories, code review, dependency scanning, secret detection, release management, and vulnerability response all matter. A skill that contains code should be scanned like code. A skill that has dependencies should be managed like a package. A skill that is distributed to employees should have a clear owner and version.

But that is not enough, because skills contain more than code. They contain behavior expressed in natural language. They contain context assumptions. They can influence an AI system at runtime. They can make tool use appear legitimate while shifting the purpose or scope of the action.

Traditional access control also needs to be extended. It is not enough to approve a skill once and assume every use is safe. A skill should grant capability, not authority. Authority must be evaluated at execution time. The same skill may be safe for one user, one purpose, one dataset, and one output, but unsafe in a different context.

The control point therefore moves closer to execution. The enterprise must evaluate who is using the skill, what purpose they declare, which data is involved, which tools are being called, what action is requested, and whether human approval is required. This is the conceptual shift from static permission to policy-based execution control.

This also means governance cannot live inside the prompt. A prompt can state rules, but it should not be the enforcement mechanism. Enforcement belongs in the platform, the runtime, the gateway, the policy decision layer, and the audit layer. The prompt may guide behavior. The platform must constrain behavior.

Governance model: skills as governed enterprise artifacts

A practical governance model should treat skills as enterprise artifacts with a lifecycle. The aim is not bureaucracy. The aim is to make reuse safe enough to scale.

During **creation**, a skill should have an owner, a stated purpose, a scope, expected inputs, expected outputs, required tools, and a risk classification. This is the moment to make implicit assumptions explicit. A skill that summarizes public documents is not the same as a skill that retrieves customer data, calls a finance system, or prepares regulatory reporting.

During **review**, the organization should inspect both technical and behavioral risk. Code should be scanned. Dependencies should be checked. Secrets should be detected. Prompts should be reviewed for unsafe instructions, overbroad behavior, hidden assumptions, and misuse scenarios. Example data should be checked for sensitivity. Tool permissions should be assessed against the intended purpose.

During **packaging**, the skill should become a versioned artifact. It should be documented, signed or otherwise traceable, and prepared for controlled distribution. The enterprise should know which version is approved, who approved it, and what changed between versions.

During **publishing**, the skill should be made available through an internal registry or catalog, not primarily through copy-paste in chat channels. The registry should distinguish experimental, approved, restricted, deprecated, and blocked skills. It should also make ownership visible.

During **execution**, the skill should pass through a runtime control layer. This layer evaluates identity, purpose, data sensitivity, requested action, tool permissions, and the need for human approval. The decision should not be based only on whether the skill is approved. It should be based on whether this execution is allowed.

During **monitoring**, the enterprise should log skill usage, tool calls, policy decisions, blocked actions, exceptions, failures, and approvals. This creates accountability and makes adoption measurable. It also enables incident response when a skill is found to be unsafe.

The governance model therefore combines software lifecycle discipline with AI-specific runtime governance. That combination is the critical point.

Reference architecture: controlled skill lifecycle and runtime enforcement

A governed skill platform resembles a CI/CD setup, but it is not only a CI/CD setup. CI/CD is sufficient to control build and distribution. It is not sufficient to control AI-mediated behavior at runtime. The architecture therefore needs two connected control planes: one for building and publishing skills, and one for executing them safely.

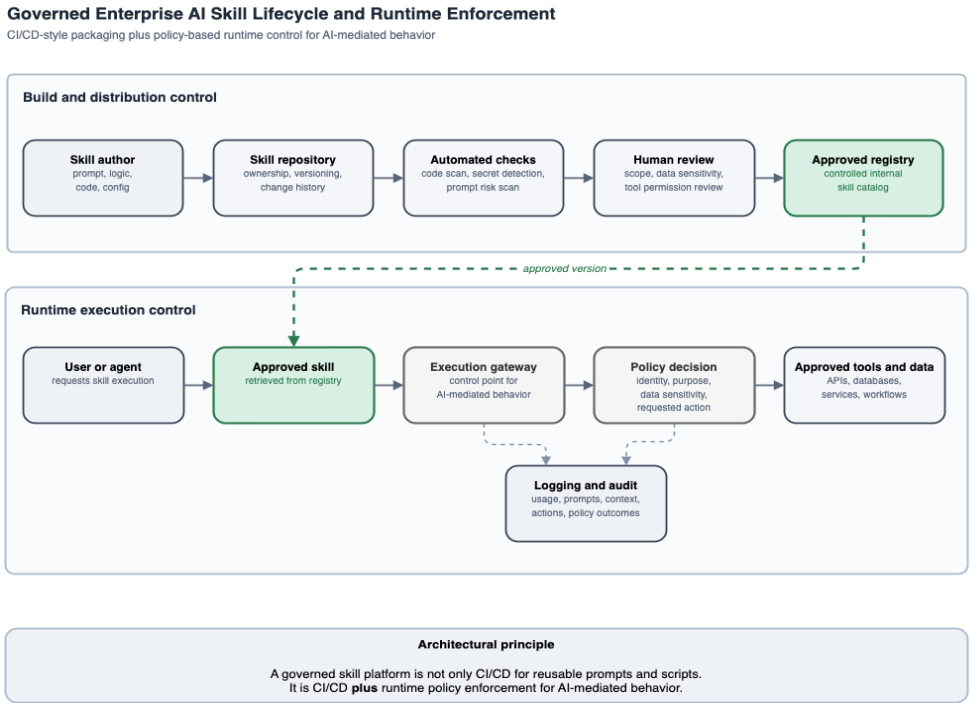

Figure 1. The governed skill lifecycle: build and distribution control (top) flows into runtime execution control (bottom). Build-time approval reduces risk, but runtime policy enforcement controls impact. Both are required.

Build and distribution control

The build lane runs across five stages. It begins with the **skill author**, who packages the prompt, logic, code, and configuration into a skill artifact. The artifact enters a **skill repository** where ownership, versioning, and change history are tracked. **Automated checks** then run: code and dependency scanning, secret detection, and prompt risk analysis. A **human review** step follows, covering scope, data sensitivity, and tool permission review - not all risks are syntactic; some are contextual, organizational, or process-related. Skills that pass review enter the **approved registry**, a controlled internal catalog of validated and versioned skills. Only versions that have completed this lane are available to the runtime.

Runtime execution control

The runtime lane activates when a skill is invoked. The **user or agent** requests skill execution. The **approved skill** is retrieved from the registry - only approved versions are accessible. The request then passes through the **execution gateway**, the central control point for AI-mediated behavior: it applies input and output filtering, mediates tool access, and routes the request to the policy layer. The **policy decision** node evaluates the full execution context: identity, purpose, data sensitivity, and the specific action being requested. Only if policy allows does execution proceed to **approved tools and data**, which exposes only the APIs, databases, services, and workflows permitted for this specific execution. All decisions and actions are continuously captured in **logging and audit**.

The connecting principle

The dashed arrow from the approved registry to the approved skill is the governance handoff: a skill approved at build time becomes available for runtime use, but build-time approval alone does not authorize execution. The policy decision layer re-evaluates each execution in context.

This architecture reflects the principle shown at the bottom of the diagram: **build-time approval reduces risk, but runtime policy enforcement controls impact. Both are required.**

Practical adoption path

The enterprise does not need a perfect framework before acting. It should begin by recognizing that unmanaged skill sharing is not a harmless productivity habit. High-risk skills should not be shared through chat or documents as the primary distribution channel. Skills that access internal systems, process sensitive data, contain executable code, or trigger business actions should move into a controlled lifecycle.

A practical first step is to create a minimal internal registry. It does not need to be sophisticated at the start. It should record the skill owner, purpose, version, risk classification, approved users, required tools, and review status. This alone creates visibility where none exists.

The next step is to introduce basic checks. Secret detection and code scanning are obvious. Prompt and context review should be added, even if initially performed by trained reviewers rather than automated tools. The review should ask whether the skill contains hidden instructions, overbroad behavior, real data examples, unsafe tool assumptions, or unclear accountability.

The third step is to classify skills by execution risk. A skill that reformats text does not need the same control as a skill that queries financial data or sends instructions to a workflow system. Risk classification allows governance to be proportionate. Low-risk skills can move quickly. High-risk skills require stronger review, runtime controls, and audit.

The final step is to move enforcement into the runtime. The organization should progressively route high-impact skill execution through policy checks. Where the action involves sensitive data, external

communication, financial impact, regulatory exposure, or changes to systems of record, execution should be constrained by policy and, where needed, human approval.

This is the same lesson enterprises learned with APIs, cloud infrastructure, data platforms, and software packages. Reuse creates scale. Scale requires governance.

Conclusion

AI skills will spread because they are useful. They let people package know-how, reduce repetitive work, and make AI assistants more effective. That value should not be ignored. The risk begins when skills move from personal productivity to shared enterprise execution without a control model.

The core issue is not the word "skill". It is the fact that enterprises are starting to distribute reusable behavior into systems that can act on data, tools, and workflows. Some of that behavior is written as code. Some of it is written as natural language. Some of it is embedded in examples and context. All of it can influence what an AI system does.

The response should not be to ban skills. The response should be to govern them as reusable enterprise behavior. That means ownership, versioning, review, packaging, controlled distribution, runtime policy enforcement, and auditability.

The future control point is not only the application, the API, or the data platform. It is the execution of AI-mediated behavior. Skills make that control point visible. Governance must follow.

References

[1] OWASP, "LLM01:2025 Prompt Injection", OWASP GenAI Security Project. Available at: https://genai.owasp.org/llmrisk/llm01-prompt-injection/

[2] OWASP, "Top 10 for Large Language Model Applications", OWASP. Available at: https://owasp.org/www-project-top-10-for-large-language-model-applications/

[3] OWASP, "Agentic Skills Top 10", GitHub project. Available at: https://github.com/OWASP/www-project-agentic-skills-top-10

[4] Ken Huang, "agentic-skills-top-10", GitHub repository. Available at: https://github.com/kenhuangus/agentic-skills-top-10

[5] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)", NIST AI 100-1, 2023. Available at: https://www.nist.gov/itl/ai-risk-management-framework

[6] Cloud Security Alliance, "Agentic AI Threat Modeling Framework: MAESTRO", 2025. Available at: https://cloudsecurityalliance.org/blog/2025/02/06/agentic-ai-threat-modeling-framework-maestro

[7] Cloud Security Alliance, "Welcome to MAESTRO", CSA Labs. Available at: https://labs.cloudsecurityalliance.org/maestro/

[8] UK National Cyber Security Centre, "Prompt injection and LLMs", referenced in public reporting and security community discussion. See also NCSC guidance and commentary on treating LLMs as inherently confusable deputies.

About this paper

This paper is published on insidin.com and is part of the Insidin knowledge base on data, AI, and enterprise governance.

License

Get in touch

Jan Uyttenhove | jan@insidin.com | linkedin.com/in/januyttenhove | insidin.com