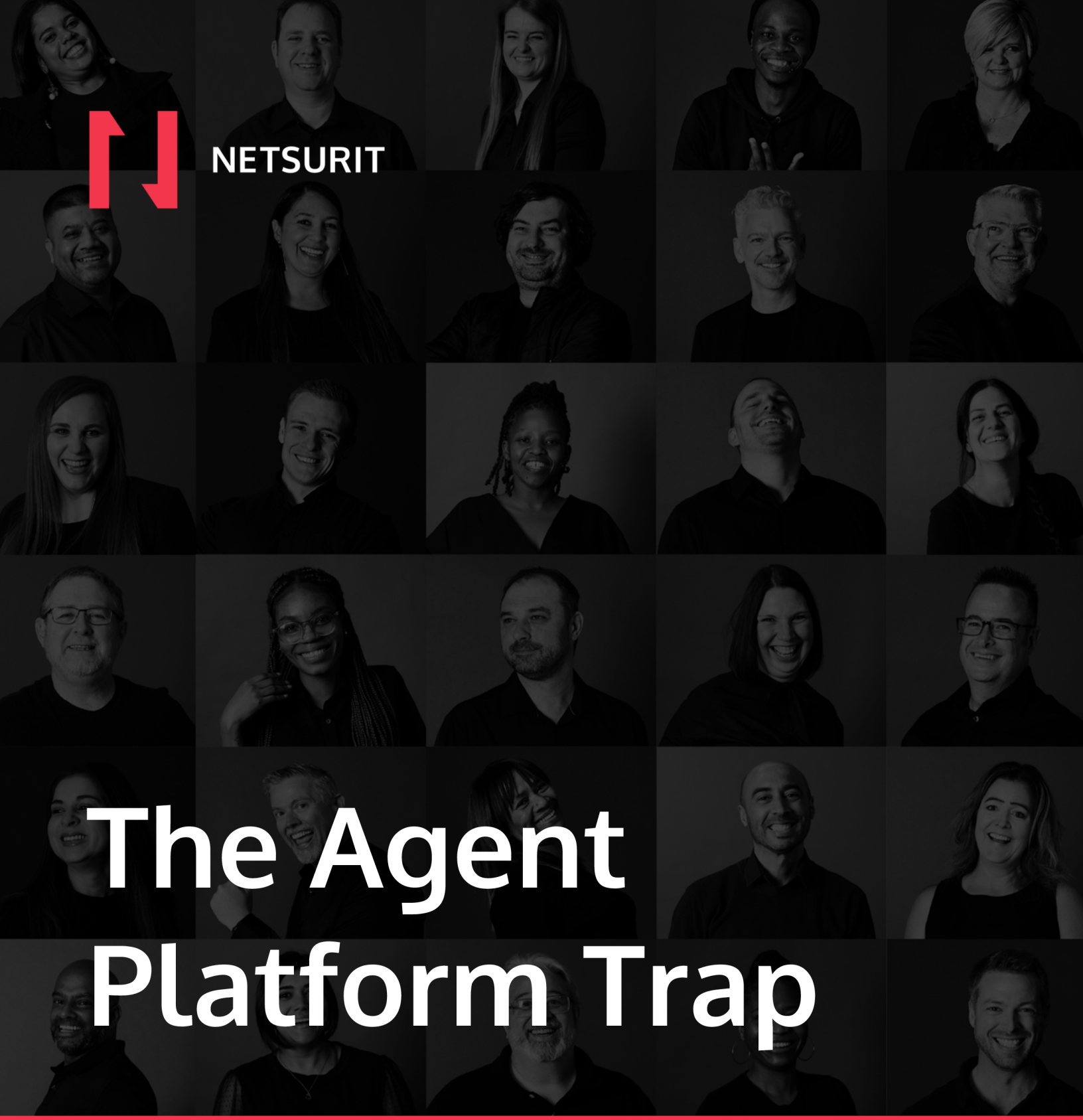


NETSURIT

The Agent Platform Trap

How smart companies make shortsighted AI decisions that are expensive to undo

by Louis de Klerk - Chief Information Officer

June 2026

Introduction

Every software vendor now claims to offer AI agents.

A chatbot that answers common questions is called an agent. A workflow that routes forms between systems is called an agent. A coding system that can investigate a bug, write a fix, run tests, and open a pull request is also called an agent. So is a customer-service bot that pulls account data and drafts a response.

These are not the same thing.

For CEOs, this creates a capital-allocation problem. For CTOs, it creates an architecture problem. In both cases, **the danger is the same: making a decision that looks sensible in the first 90 days,** only to discover 12 to 18 months later that the platform you standardized on cannot support the work that matters most.

That is the real trap in enterprise AI today.

Not buying the wrong model. Not choosing the wrong vendor slogan. Choosing an architecture whose limits are invisible at the start and costly to change later.

This matters because AI platforms are not just software purchases. They shape how work gets redesigned, how knowledge gets encoded, how governance gets enforced, what kinds of tasks can be automated, and how easily the organization can improve over time.

Early choices harden into operating assumptions. Teams build prompts, policies, workflows, integrations, and habits around them. By the time the ceiling is obvious, migration is no longer a product swap. It is an organizational change program.

The Market's Biggest Mistake

Treating all agents as points on one ladder

Most discussions of AI maturity assume a ladder. Chatbot at the bottom. Assistant above that. Agent above that. Multi-agent system at the top. That model is simple, and wrong.

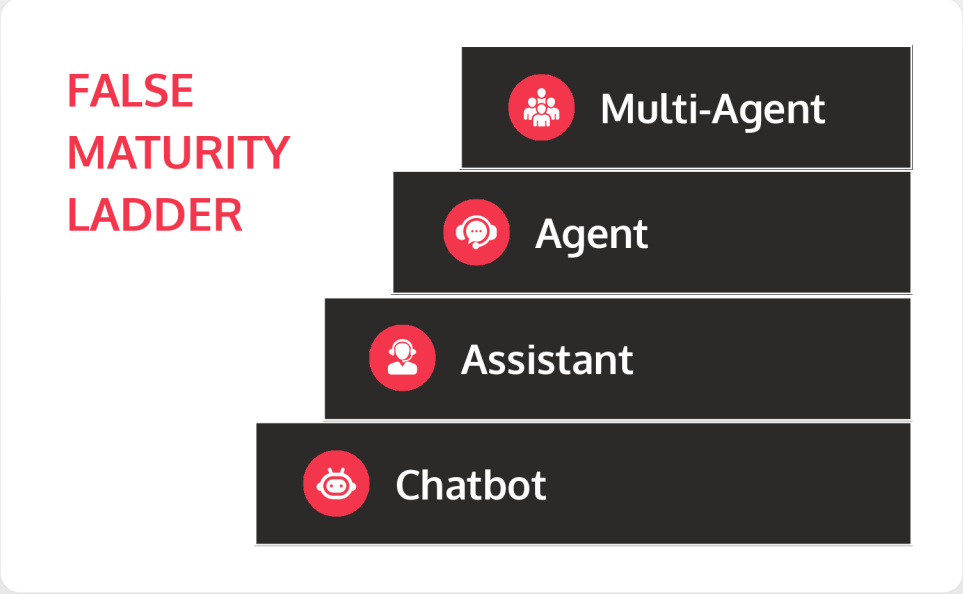
A shallow multi-agent system can be less useful than a single deep agent. A heavily governed assistant can be more valuable than a highly autonomous agent in a regulated workflow. A platform mediocre at autonomous execution may still be excellent at policy-bound service operations.

Capability is not one dimension. It is a set of tradeoffs.

If you collapse them into a single maturity ladder, you ask the wrong buying questions.

Once you stop asking "which platform is most advanced?" and start asking "which platform fits our work, our controls, and our likely future needs?" the confusion clears up quickly.

? **That is the question that actually matters.**



The LLM Model Is Not The Agent

The industry fixates on LLM models:

- Benchmark comparisons.
- Pricing per token.
- GPT versus Claude versus Gemini.

This focus is understandable but misplaced.

Andrew Ng demonstrated that GPT-3.5 wrapped in an agentic workflow outperformed GPT-4 operating alone. The infrastructure around the model mattered more than the model's raw intelligence. OpenAI's engineering team documented a three-person team producing a million-line codebase using their Codex agent, and when something failed, the fix was never "try a better model." **The engineers asked: what capability is missing from the harness?**

The harness is everything surrounding the LLM model. How context is managed and compacted as conversations grow long. Which tools are available and how they are invoked. How errors are recovered from. How state persists across sessions. How organizational knowledge is loaded and referenced. How the agent interacts with its environment. How much autonomy it is granted.

The LLM model provides raw reasoning. The harness determines whether that reasoning translates into useful action.

This distinction is not theoretical. In March 2026, Microsoft announced that M365 Copilot would route certain tasks to Claude alongside its own models. Many observers interpreted this as a significant upgrade. But the model is only one component.

If Claude's reasoning runs through the same old harness — with its tool-call-mediated execution pattern, its shallow instruction format, its rigid connectors — the output will be constrained by the harness, not elevated by the model.

The LLM Model Is Not The Agent

Putting a better engine into a car with a limited transmission does not make the car faster. **It makes the transmission the bottleneck.**

The same model running inside different harnesses produces fundamentally different capabilities. That is why this guide focuses on harness characteristics — execution architecture, skill infrastructure, protocol support, governance controls — rather than model comparisons. Models improve on commodity curves. Harnesses are architectural choices that determine ceilings.

Upgrading to the latest LLM model is easy — upgrading your harness is hard, and often expensive and painful.

The Hidden Architectural Divide Most Buyers Miss

A great deal of the market can be understood **through one underlying distinction in how the agent does its work.**

In one pattern, which we might call **mediated execution,** the agent operates by calling approved tools and receiving results back as text. Every tool definition takes up significant chunks of memory (Context Window, which is limited). It asks for something, gets a result, reasons over that result, and asks again. Every action is a round trip. Every result enters the model's working memory — which is limited (context limit). The agent is bounded by the tools it has been given and the way those tools return information. The effect is that mediated execution agents are often limited in the number of tool calls they can make per conversation, making tasks like "Summarize my 100 meetings this month" impossible to complete.

In the other pattern, which we might call **environmental execution,** the agent has its own persistent working environment. It can read and write files, execute code, install packages, run tests, inspect outputs, and iterate — all within a continuous workspace. It does not route every action through a narrow tool interface. It operates more like a person at a computer than a caller on a phone. The difference is that what has to be included in memory (context window) is significantly reduced (95%) — making long running tasks or tasks that needs to work across a large number of items possible — e.g. "Summarize the 500 meetings my team had with Customer X" is not only possible, but much cheaper to run (fewer tokens).

This distinction is often treated as a technical detail. It is not. It has direct strategic implications.

The Hidden Architectural Divide Most Buyers Miss

Anthropic's engineering team quantified the difference. On agentic research benchmarks, letting the agent write code that orchestrates multiple tools — rather than making individual tool calls — was the single factor that fully unlocked agent performance. A task requiring 20 tool round trips, each loading thousands of data points into context, collapses to one script that returns a three-line summary. This impacts not only the costs involved, but also factors like speed, intelligence of the agent, and the space left in memory for the agent to do the work asked for. **Mediated execution used 150K tokens compared with 2K tokens with environmental execution** — a vast difference.
***See Appendix A for more details.**

However the mediated pattern tends to be easier to govern. It is often better aligned with enterprise auditability, role-based access, and bounded actions. It can be the right choice when work must be tightly controlled, especially in regulated environments.

The environmental pattern has a higher capability ceiling. It is better suited to long-running tasks, deeper synthesis, messy exceptions, and working with large bodies of organizational knowledge. It is stronger where value comes from adaptation, not just compliance with prebuilt flows.

Neither is inherently better. But they are not interchangeable. And the strategic mistake is assuming you can buy the more constrained pattern for convenience and later grow into the more capable one without pain. Moving between them changes the governance model, the security posture, the operator workflow, and the way teams structure work itself.

The Three Practical Classes Of Work That Matter Most

Most organizations do not need **one generic "agent strategy."** They need to understand the kinds of work they are trying to support. **Three broad classes matter most.**

1. **Structured workflow work** is operational work with defined steps, known systems, clear approvals, and bounded actions. Service desk operations, case routing, access requests, HR operations, order management, policy-bound internal processes.

The primary challenge here is not intelligence. **It is control.** The system must be reliable, auditable, bounded, and well-integrated with systems of record. A highly flexible agent may be less valuable than a more constrained one if the constrained one can be trusted, monitored, and approved.

2. **Unstructured knowledge work** is defined by ambiguity, synthesis, judgment, and exception handling. Investment analysis, due diligence, policy analysis, underwriting support, market research, document-heavy advisory work.

The challenge here is reasoning across incomplete, conflicting, or fragmented information. These tasks involve many documents, sources, and judgment calls. They benefit from agents that can plan, iterate, compare, refine, and adapt.

3. **Cross-source synthesis and profile-building** is the category most buyers underestimate, and for many professional-services firms it is the most important one. This is the work of assembling and maintaining a living understanding of a client, company, household, or case from scattered signals across many systems.

The Three Practical Classes Of Work That Matter Most

Think of a firm that wants to build and continuously update a client profile using emails, meeting notes, CRM records, invoices, account statements, prior recommendations, public filings, market data, and regulatory constraints. That profile then supports a recommendation, identifies risks, or guides a next-best action.

> **Shallow assistants can retrieve fragments from these systems, but they are much less reliable or capable at building, maintaining, and reasoning over a durable picture that changes over time. For firms in tax, accounting, wealth management, and private equity, this use case is not peripheral. It is central.**

Skills: The Real Compounding Asset

Models improve quickly. Vendor packaging changes. Connectors evolve. **None of those are likely to be your durable advantage.**

What will matter far more is the quality of the organizational knowledge you encode and improve.

That knowledge takes many forms: review checklists, exception rules, recommendation criteria, client-service playbooks, communication templates, compliance guardrails, evidence standards, escalation rules, quality criteria, and prioritization heuristics. Some of it is formal. Much of it is tacit — living in the heads of senior professionals, in habits, in old memos, in operating lore.

In practical terms, this knowledge can be captured as **skills: curated instruction sets that encode how your organization does specific work.** A skill might be a file containing your security review checklist, your change management procedure, or your underwriting criteria, with references to detailed sub-files that contain the specifics.

The most effective skill architectures use progressive loading. The top-level file contains general instructions and references to detailed sub-files. A security review skill might reference separate criteria documents for PCI-DSS, SOC 2, and HIPAA. The agent reads the top-level instructions, then loads only the sub-files relevant to the current task. The working memory stays clean, and a single skill can encode enormous depth without overwhelming the agent with irrelevant detail in its memory (context window).

Progressive loading works dramatically better in environmental-execution systems. The agent has a file system. Loading a sub-file is a local file read — instant and free. The agent can search across 50 skill files to find the relevant section, read just that section, write intermediate notes to a scratch file, and keep only conclusions in working memory. **The file system functions as extended memory.**

systems. The agent has a file system. Loading a sub-file is a local file read — instant and free. The agent can search across 50 skill files to find the relevant section, read just that section, write intermediate notes to a scratch file, and keep only conclusions in working memory. **The file system functions as extended memory.**

In mediated-execution systems, every sub-file load is a tool call requiring a round trip. The loaded content enters working memory in full. The agent cannot selectively load three relevant paragraphs from a 20-page reference. This means a mediated agent with a large skill library faces an unsolvable tradeoff: load too little and miss critical context, or load too much and crowd out everything else. This tradeoff gets worse as the skill library grows.

The most advanced skills contain executable code — scripts, validation routines, and procedural logic the agent can run, not just read. An environmental-execution agent can execute these directly. A mediated agent cannot, but has to write the code again and send it to a code execution container using a tool call — using precious memory (context window). This is fine for a few code executions, but quickly runs into context window limits when looping over items.

This is why the architecture decision and the skills decision are inseparable. **The ceiling for skills investment is structurally higher on environmental-execution platforms,** and the gap widens as the organization's skill library matures. A firm that steadily converts operational knowledge into reusable, versioned, measurable skills will improve faster than one that turns on generic AI features and hopes for the best.

Why Control Matters More Than Capability-First Narratives Admit

Lets turn on the brakes and look at the bigger picture, before drawing the conclusion that all that matters is the mediated vs environmental execution pattern. This is a common mistake in AI strategy: assuming the most capable platform is the obvious long-term winner.

That view is incomplete.

In many real businesses, especially regulated or policy-constrained ones, the most important question is not "What can the agent do?" but "What can the organization safely allow it to do?"

A platform that is less flexible but easier to audit can be the right choice. A system that requires human approval at defined checkpoints may create more value than a more autonomous one that cannot pass compliance review.

But this should not be misunderstood. **The goal is not low capability. The goal is governed capability.** A weak architecture is not automatically a safe one. If it relies on vague prompts, inconsistent retrieval, poorly versioned instructions, and limited traceability, it may be both less capable and still hard to audit.

What organizations need is a combination of clear control points, versioned instructions and policy content, traceability of inputs and outputs, bounded permissions, human approval where required, and monitoring with escalation paths. If those controls can be implemented credibly around a stronger agent, then a higher-ceiling platform may still be the better strategic choice.

The question is not capability versus control, but how much capability we can actually govern.

The Competitive Threat From Learning Loops

The biggest strategic threat from AI-first competitors is not that they have access to a better model. Base models are available to everyone.

The threat is that they build learning loops into how the company operates — they build systems that learn, adapt, and improve as they are used — codified business processes with evolving memory under human oversight.

They let agents handle work, measure performance, capture exceptions, identify where outputs failed, update skills, test the changes, and feed improvements into the next cycle. **They do not just use AI. They use AI in self-improving loops.**

> **A traditional firm may use AI mainly as a productivity feature: draft faster, summarize faster, search faster. Helpful, but incremental.**

Then AI-first firm treats each client interaction, exception, review cycle, and missed recommendation as data for skill improvement. Over time, it builds better profile synthesis, better issue detection, better escalation logic, and better encoded judgment. The gap may not look dramatic in month one. By month eighteen, it can be material.

This is why short-sighted platform choices are dangerous. If your platform supports only isolated prompts or shallow assistant behavior, while competitors are building closed-loop systems that improve from outcomes, you are not just missing features. **You are missing a different rate of organizational learning.**

What Changes For SMB And Mid-Market Firms

Most AI strategy writing assumes a large enterprise buyer with broad architecture teams, deep governance capacity, and room for experimentation.

Many real buyers are not like that.

In the SMB and mid-market — especially in sectors like tax, accounting, private equity, and wealth management — **the constraints are different.** Smaller IT teams face higher sensitivity to implementation burden. Serious compliance requirements without enterprise-scale governance infrastructure. Little appetite for multi-year platform rationalization.

For these firms, the main risk is not always choosing too low a capability ceiling. It can also be choosing a platform that is **too heavy, too custom, or too dependent on talent the organization does not have.**

Three things matter especially:

1. The platform must fit the firm's actual operating capacity.
2. It must match the shape of the firm's high-value work — a wealth management firm doing relationship-centric synthesis has different needs from a service organization automating ticket flows.
3. The knowledge assets the firm builds should remain portable.

For this segment, the winning move is rarely "buy the most advanced architecture" or "buy the most familiar suite." **It is: choose a platform that can improve a small number of high-value workflows quickly**, within your actual compliance and operating capacity, while keeping your knowledge from becoming trapped in proprietary configurations.

How CEOs And CTOs Should Actually Prioritize

When leaders choose among platforms, they often ask the wrong top-level question — which vendor is best, safest, or most advanced.

The better set of criteria is:

1. **Identify the work that matters**

   Map 10 to 20 of your most valuable workflows and classify them: structured workflow execution, unstructured knowledge work, cross-source synthesis and individual vs team work.

   That introspection alone clarifies more than any vendor demo that leads the witness.

2. **Define the control floor**

   What controls are non-negotiable? Privacy boundaries, role-based access, approval requirements, logging, retention. Any platform that cannot meet this floor **should be eliminated regardless of its capability ceiling.**

3. **Define the 24-month headroom requirement**

   If the roadmap includes only bounded workflow support, a more controlled platform may be exactly right. If it includes richer synthesis, longer-running tasks, reusable skills, and more ambitious recommendation support, the platform needs environmental execution, progressive skill loading, and protocol-based connectivity. The goal is not to buy for some hypothetical far future. It is to avoid choosing a system that tops out exactly where the next real wave of value (to you) begins.

4. **Assess operating burden honestly**

   A platform requiring highly specialized implementation or extensive custom operational scaffolding may be a bad choice for a lean firm even if its raw capability is attractive.

How CEOs And CTOs Should Actually Prioritize

5 Keep assets portable

Keep skills, policies, evaluation criteria, and business logic in formats that can move. Use open protocols — MCP for tool connectivity, A2A for agent interoperability — where available. The more your advantage depends on organizational knowledge, the less you want it buried inside opaque configuration surfaces.

6 Start building skills immediately

Interview the people who do the work best. Capture rules, edge cases, and quality criteria — the things not in any manual — then version control them, assign ownership, and review them quarterly. Measure agent performance per skill. This is the one investment that compounds regardless of which platform runs it.

The Practical Tradeoff

Controlled room to grow

If there is one principle that captures the right approach, it is this: choose platforms that give you controlled room to grow.

That means: **do not choose a platform that cannot meet your control requirements.** Do not choose one whose capability ceiling is clearly below your likely next wave of use cases. **Deploy conservatively at first.** Keep humans in the loop where stakes are high. Build reusable knowledge assets from the start. Learn from exceptions and improve systematically.

Also keep in mind that you don't have to limit yourself to just one platform. Balancing other metrics like cost, complexity, user adoption and compliance — all alongside capability often means that having multiple platforms is often the best bet.

One platform could target personal productivity, a different platform could target team work, and another platform could target processes.

Decision makers need to try to avoid these mistakes:

1. **Buying only for control and governance,** then discovering too late that the platform cannot support the work that differentiates the business.

2. **Buying only for capability,** then finding that nothing reaches production because the organization cannot trust, govern, or operationalize it.

3. **Buying for lowest cost and familiarity,** then finding that you have neither the capability you need, nor the governance and control demanded.

The Practical Tradeoff

Controlled room to grow

The true cost of a short-sighted platform decision is not only wasted spend.

It is slower learning, weaker workflow redesign, trapped knowledge, lower ambition, and a harder migration when the business finally realizes it needs more than the platform can give.

That is the corner companies paint themselves into. Not by making an irrational choice, but by making a locally rational choice without a clear view of the tradeoffs underneath it.

> **The leaders who make good decisions here will not be the ones who chase the most impressive demo. They will be the ones who understand which tradeoffs matter, which limitations are temporary, which are structural, and which choices create optionality instead of eroding it.**

by Louis de Klerk, CIO

Netsurit
June 2026

NETSURIT

Appendix A

Mediated Execution vs Environmental Execution

Here are some concrete differences between the different styles of execution.

Dimension	Mediated execution (direct MCP)	Environmental execution (code-in-sandbox)
Token usage (complex workflow)	~150,000 tokens — all tool definitions + every intermediate result passed through context.	~2,000 tokens — 98.7% reduction. Only relevant tool signatures loaded on demand; data stays in execution environment.
Tool definition loading	All definitions loaded upfront into context (~15,400 tokens per call across 63 tools in one study).	Progressive disclosure via filesystem — agent reads only the tool files it needs for the current task.
Input tokens (independent replication)	771,000 tokens across a multiquery benchmark.	165,000 tokens — 78.5% fewer. Consistent direction with Anthropic's finding, lower magnitude due to different task mix.
Intermediate data handling	Every result routed back through the model — a 2-hour transcript can add 50,000+ tokens and may be processed twice.	Results processed inside the execution environment; only summaries or filtered outputs return to the model.
Control flow (loops, conditionals)	Model must reason through each iteration — each cycle is a round-trip through context, adding latency and tokens.	Written as real code, executed by the runtime — no model round-trip per iteration, no wait per branch.

Appendix A

Mediated Execution vs Environmental Execution

Here are some concrete differences between the different styles of execution.

Dimension	Mediated execution (direct MCP)	Environmental execution (code-in-sandbox)
Context window ceiling	Hard cap — large documents or many tools can exhaust the window and break the workflow entirely.	Practically removed as a constraint — large datasets stay in the execution environment; context holds only what the model needs to reason about.
Privacy / data exposure	All intermediate data — including PII — passes through the model context.	PII can be tokenized by the harness before reaching the model; real values flow tool-to-tool without touching context.
Infrastructure overhead	Low — no execution environment needed; tool calls handled directly by MCP client.	Higher — requires sandboxed execution environment, resource limits, and monitoring.

Sources: Anthropic Engineering (Nov 2025), AIMultiple replication (Jan 2026).

Appendix B

Vendor Comparison

Here are some concrete differences between the different styles of execution.

Vendor / product	Dominant harness style	Control posture	Capability ceiling
Microsoft Copilot Studio / M365 Copilot agents	Mediated execution	High	Low-Medium
Microsoft Azure AI Foundry Agents	Mediated execution	High	Medium
Anthropic Claude Code	Environmental execution	Medium-High	High
OpenAI Codex / Codex CLI	Environmental execution	Medium	High
n8n AI Agent / agent workflows	Mediated execution	Medium-High	Medium-High
Hatz	Mediated execution	Medium	Low