workato

MCP DESIGN SERIES

The Field Guide to MCP Server Design

What we learned building 25+ enterprise-ready MCP servers, from design principles to individual tool patterns. Fours parts, one through-line: **build for the model doing the reasoning, not the developer building the system.**

PART 01

Design 101

The fastest way to build a confusing MCP Server is to start with the application. Here's the design order that actually works.

PART 02

A Mental Model

Inconsistent MCP behavior is almost never a model problem. It's a design problem — and it starts with the wrong mental model.

PART 03

First Principles

The MCP Servers that work consistently follow four principles. The ones that don't violate them, often in subtle ways.

PART 04

Building for LLMs

You can have the right scope, the right principles — and still ship tools that break. Here's what actually fails, and how to fix it.

workato

HOW WE MCP // PART 1 OF 4

Design 101

The fastest way to build a confusing MCP Server is to start with the application. Here's the design order that actually works.

The Shift

What can we expose? --> Who needs it, and to do what?

LLMs operate on: Intent --> Action --> Outcome

Your server should map to this—not to an API schema

When you start with the application...

You end up with

- Multiple unrelated personas in one server
- Competing workflows with no clear priority
- A growing list of "might be useful" tools
- Ambiguous tool names that overlap in scope

The same request maps to **multiple valid tools.** The LLM has to guess which path is correct—and behavior becomes inconsistent. The LLM isn't confused. **The design is.**

Good MCP design removes ambiguity. Bad MCP design creates it.

Design in this order...

1. **Define the persona**

   Who is this for? If the answer is "multiple unrelated roles", that's your first red flag. **One server, one primary user.**

2. **Define the core user cases**

   Not "manage customers", but **"investigate a billing issue" or "update deal stages after a call."** Specific, completable actions.

3. **Define the workflow boundary**

   What does "done" look like? A good server lets the LLM complete a **meaningful unit of work ent-to-end** without bouncing across unrelated concepts.

4. **Now design your tools**

   Once persona, use cases, and workflows are clear, the right tools become obvious. **Tools aren't the starting point. They're the outcome.**

3 questions before you write a tool

- [ ] Can I describe this server in one sentence?
- [ ] Is it clearly for one persona (or related ones)?
- [ ] Can a core use case complete ent-to-end within this server?

If any answer is "no", you don't have a tool problem. You have a scope problem.

workato

HOW WE MCP | PART 2 OF 4

A Mental Model

Inconsistent MCP behavior is almost never a model problem. It's a design problem — and it starts with the wrong mental model.

APIs are designed for **EXECUTION** vs MCP Servers must support **REASONING BEFORE EXECUTION**

THE BROKEN ASSUMPTION

API design patterns that fail LLMs

- Mirror the API surface
- Expose flexible operations
- Rely on implicit behavior
- Assume the consumer will adapt

BEFORE CALLING ANY TOOL, THE LLM MUST...

01 Decide whether a tool should be used
02 Choose which tool applies
03 Determine how to populate inputs
04 Predict what will happen
05 Interpret the result
06 Decide what to do next

WHAT AN LLM ACTUALLY SEES

Its entire world:

- Tool names
- Descriptions
- Parameters
- Outcomes

If something isn't made explicit here, it effectively doesn't exist.

4 QUESTIONS EVERY TOOL MUST ANSWER

- [ ] Can the LLM decide when to use this?
- [ ] Can it distinguish this from other options?
- [ ] Can it predict what will happen?
- [ ] Can it understand the result well enough to continue?

If your MCP Server feels unpredictable, ask: was this designed for execution — or for reasoning?

workato

HOW WE MCP | PART 3 OF 4

First *Principles*

The MCP Servers that work consistently follow four principles.
The ones that don't violate them, often in subtle ways.

01	02	03	04
CLARITY over completeness	**COHESION** over convenience	**EXPLICITNESS** over inference	**DETERMINISM** over cleverness

01 Clarity over completeness

More tools don't add capability — they add confusion. When multiple tools look valid, selection becomes probabilistic. Small phrasing changes produce different choices.

BAD	GOOD
search_records	search_records
find_records	(criteria)
query_records	get_record (by ID)

If multiple tools look valid, the model will guess.

02 Cohesion over convenience

Mixing unrelated workflows in one server means the same request can map to multiple interpretations. The LLM has no clear context — so it guesses.

BAD—ONE SERVER	GOOD—SEPARATE
search_opportunities	Sales server
create_support_ticket	Support server
manage_users	Billing server

If a request can mean multiple things, your scope is too broad.

03 Explicitness over inference

Hidden side effects, undocumented limits, missing prerequisites — if they're not stated, the model assumes incorrectly. What's obvious to you is invisible to the model.

BAD—HIDDEN EFFECTS	GOOD—EXPLICIT
update_order(id, status)	update_order_status
+ sends email	send_order_notification
+ updates inventory	process_payment

If it's not stated, the model has to guess.

04 Determinism over cleverness

If the same inputs produce different outputs, the model can't form stable expectations. Variation is fine — hidden variation is not. Make it vary on explicit inputs, not hidden context.

BAD—HIDDEN VARIANCE	GOOD—EXPLICIT
get_customer(id)	get_customer(id, level)
fields vary by role/state	summary | standard | full

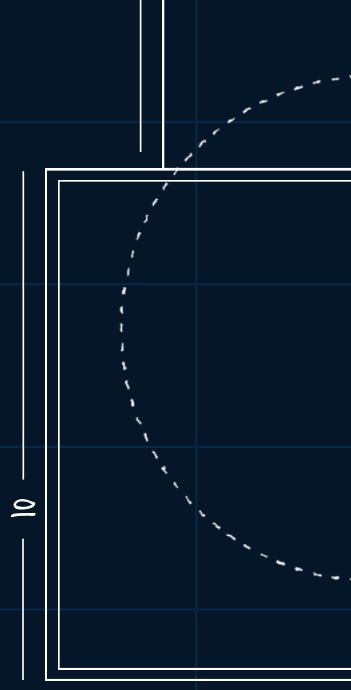
Determinism isn't removing variation, it's making variation predictable.

*All four principles do the same thing: they **reduce ambiguity.***
When ambiguity goes down, reliability goes up.

workato

HOW WE MCP | PART 3 OF 4

Building for LLMs



You can have the right scope, the right principles — and still ship tools that break. Here's what actually fails, and how to fix it.

LLMS SEE tool names	LLMS SEE descriptions	LLMS SEE parameters	LLMS SEE outputs

Hard to choose

Overlapping tool names force the model to guess between equally valid options.

BAD
```
search_customers
find_customers
```

GOOD
```
search_customers
get_customer (ID)
```

Make the right choice obvious.

Parameters don't map

Generic inputs force interpretation before action. Each parameter should match how users express intent.

BAD
```
search
(query, filters, opts)
```

GOOD
```
search
(name, date, status)
```

Reflect how people ask.

Too much data

Unbounded results bury signals and waste context. More data doesn't improve reasoning — it makes it harder.

BAD
```
returns 500 rows
no truncation
```

GOOD
```
bounded results
has_more: true
```

Return bounded, usable data.

Outputs lack meaning

Different outcomes require different decisions. Status alone isn't enough — the model needs to know why.

BAD
```
status: success
data: []
```

GOOD
```
nothing_found
permission_denied
```

Communicate meaning, not status.

Outputs aren't usable

Raw IDs and loose structure force reinterpretation before the next step. That's where errors enter.

BAD
```
{ id: "a3f9x" }
```

GOOD
```
structured fields
reusable in next call
```

Enable the next step without guessing.

BEFORE SHIPPING ANY TOOL, ASK

- [ ] Does the name make the right choice obvious?
- [ ] Do parameters map to how users express intent?
- [ ] Are results bounded with clear truncation signals?
- [ ] Do outputs distinguish meaningfully different states?
- [ ] Can the model use the output directly in the next step?

THE THROUGH-LINE

MCP tools aren't just functions. They are interfaces for reasoning.

If the model has to guess at any step — selection, input, output, or interpretation — reliability breaks. Design for the model doing the reasoning, not the developer building the system.

READY TO PUT THIS INTO PRACTICE?

Build MCP Servers that actually *work.*

Workato's pre-built MCP server catalog applies every principle in this guide — designed from the ground up for enterprise AI agents. 25+ servers, production-ready governance, and composable cross-system orchestration built in.

- **25+ pre-built servers** for Salesforce, Workday, GitHub, Gmail, Slack, Zendesk and more
- **Verified User Access (VUA)**, tool-level RBAC, and immutable audit trails
- **Composable across systems** — connect any AI agent to any business workflow
- **Deploy in minutes, not months** — fully managed by Workato

TALK TO AN EXPERT

See Workato Enterprise MCP in action

Book a 30-minute session with an MCP specialist. We'll walk through your AI use case and show you exactly how Workato's pre-built servers can get you to production faster.

Book a meeting →

workato.com/request-demo

→ **Explore the MCP server catalog**
Browse all 25+ pre-built servers with tool details