01 / PART 1 · UNDERSTAND THE DEPENDENCY

What Your AI Depends On

Why API quality shapes user, developer and agent experiences, even when the software connections are invisible.



You approve an article and ask an AI assistant to put it on your company's website. It replies, "Done." But the request has only entered a queue, and nobody can confirm whether readers can see the article. Publishing again could create a second copy. The conversation has ended; your team's uncertainty has just begun.

> **Who this paper is for—and what you will learn**
>
> For business leaders, AI practitioners and software teams building or choosing applications and agents. AI can hide the software connections an action depends on. This paper explains what APIs do, how their quality affects people, developers and agents, and what to ask before relying on them. It also introduces how APIOps Cycles helps teams agree responsibilities and improve a shared practice.

The interface people never see

An application programming interface, or API, defines how software requests information or an action from other software. A connector can hide these requests behind a screen; an AI assistant can hide them behind a conversation. AI-generated code may depend on an API too. Understanding the actual connection helps you judge what the experience relies on.

A user interface (UI) helps a person understand actions and consequences. An API needs the same design attention for the software using it. If it cannot distinguish keeping an article private from making it public, or confirm which happened, each application must resolve that uncertainty. A polished screen or fluent conversation depends on those underlying abilities.

One API may support several websites, data pipelines and agents. Unclear meanings and unsafe actions can spread through all of them at machine speed. That reach can justify as much design attention as a UI, and sometimes more—including inside your own organisation.

02 / THE API BEHIND A WEBSITE ACTION

The same task, a clearer path

A useful API helps its consumers understand the action, carry it out and establish the result. Compare two designs for one publishing task.

APIs can expose many capabilities, from checking a balance to arranging a delivery. Here, a content-management API lets software make an approved article visible on a company website. The editor's Publish button and an AI assistant's tool both call it. The API provides controlled access while hiding storage, queues and other internal details that consumers should not have to manage.

Consumer need	Supports a reliable journey	Leaves gaps for the consumer
Get started	Use limited test access, clear descriptions and working examples.	Ask a maintainer about hidden settings; borrow a shared account.
Take the right action	Name the website and approved version. The service checks access and approval.	Guess what "Save" does; risk publishing text changed since approval.
Know the result	Check whether publication is waiting, failed or complete; get the public address.	Receive "OK" for a queued request; have no way to confirm publication.
Recover safely	Understand the error. Repeat a request safely within the documented retry period.	Receive "failed" without guidance; risk a duplicate article when trying again.

Both designs may appear to work on an ordinary day. The difference becomes clear when someone first connects, an article changes or a reply is lost. Quality here means fitness for this task and its consumers, supported by clear descriptions and tested behaviour. [2, 3]

Two experiences behind the visible interface

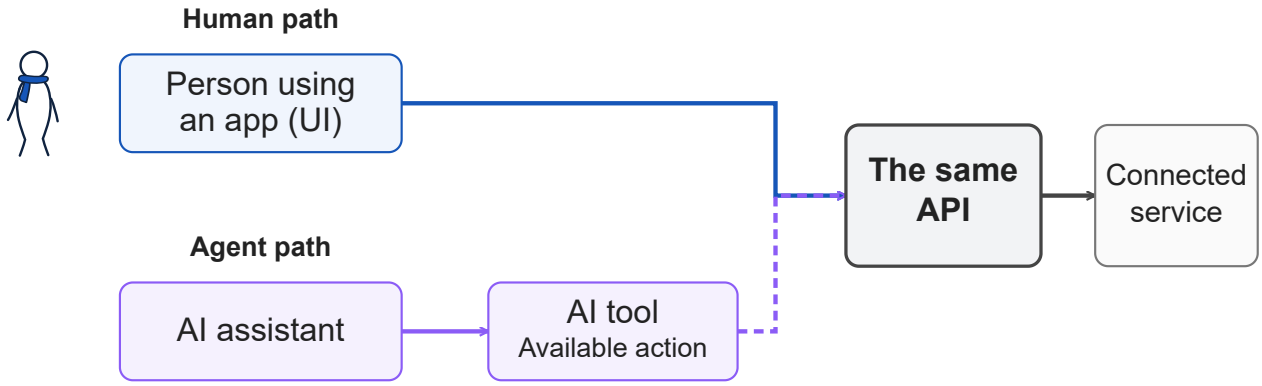
Developer experience (DX) covers finding an API, understanding its fit, getting access, testing, diagnosing problems and maintaining the connection. Clear examples and test access help developers build both the editor UI and the assistant's tool.

Agent experience (AX) concerns how an agent finds and chooses actions, supplies inputs, interprets results and recovers from problems. The tool must make those choices clear, and the API must support the promised behaviour. The next page shows how these experiences connect.

03 / USER, DEVELOPER AND AGENT EXPERIENCE

The experience your API enables

People interact with screens or conversations. Developers and agents depend on the meaning and behaviour of the software connection behind them.



Requests shown; results return along the same paths.

The human uses a UI; the agent uses a tool. Both depend on the same API. DX supports both integrations; AX concerns the agent's interaction. APIOps Cycles visual adaptation, CC-BY-SA 4.0. Model Context Protocol (MCP) is one way to expose tools to AI applications. [1]

For the person: an action with a trustworthy result

The editor needs to know which article version is public, where it appears and whether the request is still waiting. A screen can show those distinctions clearly when the API supplies the relevant result. The person should be able to act on that result with confidence.

For the developer: less meaning to reconstruct

Clear descriptions, examples, test access and useful errors help developers build the editor UI and AI tool around the same publication meanings. They can test shared behaviour and diagnose failures without reconstructing undocumented rules. AI-generated integration code needs that guidance too.

For the agent: actions it can choose and interpret

Agent experience (AX) covers finding and choosing an action, supplying inputs, interpreting results and recovering from problems. Here the assistant must distinguish saving a draft from making it public, then interpret waiting or completion correctly. Developers support that experience through tool design and integration; the application still has to use the API's capabilities correctly.

04 / WHAT AGENTS CHANGE

Familiar requirements, a different caller

Many agent requirements are established integration needs. Model-driven tool selection adds another place where meaning, permissions and results can go wrong.

MCP provides tool descriptions and structured input definitions, called schemas. They describe what can be supplied. How much the assistant may decide remains an application design choice; some AI workflows follow fixed steps. [1]



Selection
Which action?

Authority
May it act?

Results
What happened?

Validation
Does it work?

Four concerns for the whole interaction. APIOps Cycles design-system symbols, CC-BY-SA 4.0.

Selection: make the choices distinguishable

A developer normally chooses the call sequence in advance. An assistant may choose from tool descriptions while carrying out a request. “Save draft” and “publish approved version” must have clear meanings and input definitions.

Authority: enforce what the caller may do

Permissions and approval belong in application and service controls. A tool description can explain a boundary; it cannot enforce one.

Results: support a sound next decision

Return states and errors that the caller can interpret. Waiting, failure and an unknown outcome must remain distinguishable from completion. Recovery and repeated requests need defined behaviour. [1, 2]

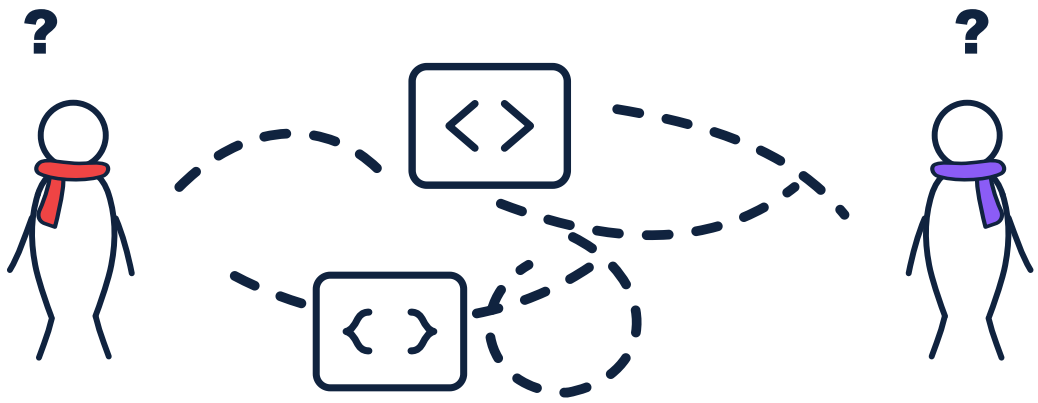
Validation: test the whole intended task

Check the API behaviour and how the application selects the action, supplies inputs and interprets results. Include ambiguity and relevant failures. Keep API, tool and application owners involved when those parts change.

05 / THE WORK BEHIND A USABLE CONNECTION

One API, different assumptions

The editor’s screen treats “OK” as “submitted”. A new assistant treats it as “published”. The same response now creates two different experiences.



People can depend on the same API while holding different assumptions. APIOps Cycles design-system scene, CC-BY-SA 4.0.

The original editor UI knew to show “Waiting” after an “OK” response. Its developers learned that rule from the API team. The team building the assistant sees “OK” and assumes publication is complete. The API still behaves as before; the unwritten rule has failed to reach a new consumer.

This can happen when an internal API grows beyond its first application. A screen hides technical settings and colleagues explain unusual results. A new assistant, data workflow or partner application brings different expectations. Each new consumer makes the need for shared meanings and responsibilities more visible.

Agree the promise and who keeps it

Bring the provider and consumer teams together around one task. Agree what the API means, what each action promises and who maintains implementation, examples, tool definitions and support. Give that continuing work time and an owner.

The business purpose and priorities (product strategy), and fit with shared systems (platform architecture), still matter. Uncertain value or untested assumptions may change or limit a trial. If a supplier owns the API, request evidence and decide which limitations your application can accept.

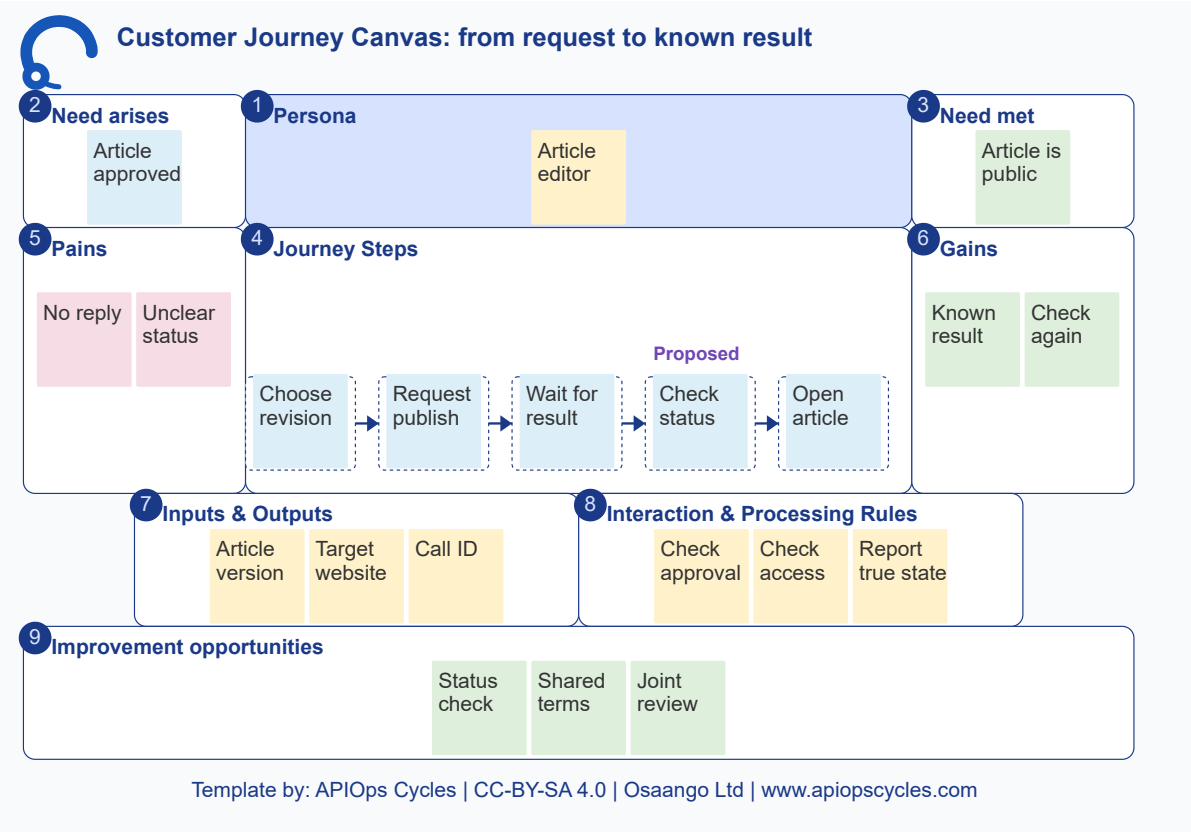
APIOps Cycles connects business purpose, consumer experience, architecture, design and ongoing improvement. Its goal is to establish the decisions and responsibilities that keep a digital capability useful as its consumers change. Teams can start with one practice while keeping wider dependencies visible. [5]

06 / FROM AN UNCERTAIN RESULT TO A CHECKABLE ONE

From “submitted” to knowing what happened

The editor’s task is the same through a conventional UI or an AI assistant.

The current journey ends in uncertainty. The canvas shows the proposed journey after the team identifies the missing status check.



Intended journey rather than observed results. APIOps Cycles / Osaango Ltd, CC-BY-SA 4.0. Circles identify fields.

The caller saves and sends a reference; the service links it to the publication operation. An authorised check returns waiting, failed or complete, with the public address on completion.

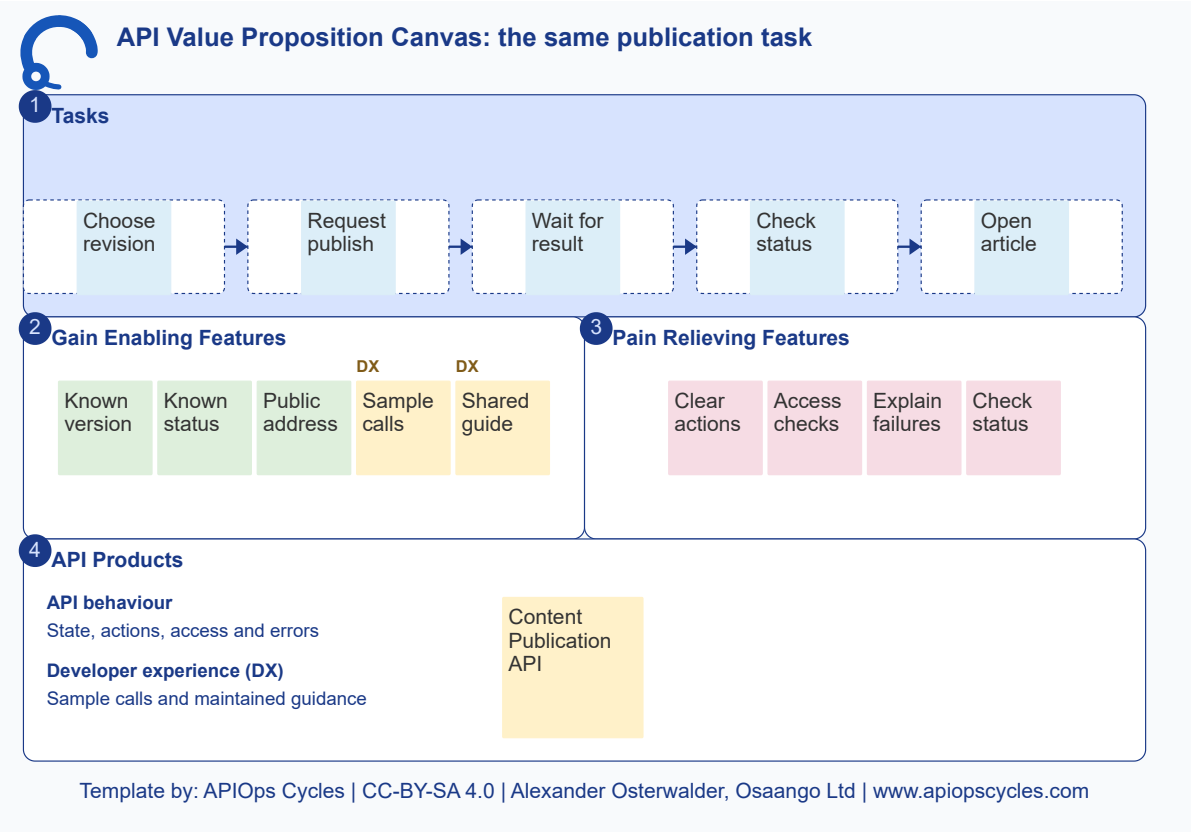
> **The team practice behind the journey**
>
> API, tool and guide owners test lost replies, unknown references and denied access.

A status check recovers visibility. Safe retries, approval and version protection need separate checks; strategy and architecture gaps remain open. [2]

07 / FIVE QUESTIONS BEFORE RELYING ON THE CONNECTION

What should your team ask next?

The Customer Journey Canvas reveals unmet needs; the API Value Proposition Canvas translates them into proposed features.



Proposed features for implementation and validation. Template: Alexander Osterwalder and Osaango Ltd / APIOps Cycles, CC-BY-SA 4.0. Circles identify fields.

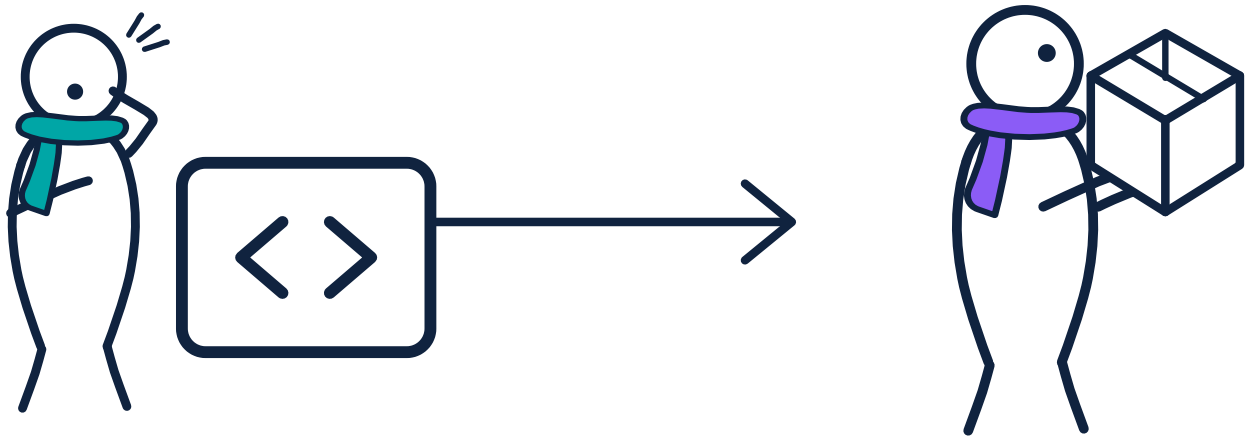
1. Can consumers understand the action?
2. Is authority enforced?
3. Can outcomes be verified?
4. Can failures be recovered from?
5. Who maintains the experience?

> **Start with one useful improvement**
>
> Improve one practice. Own strategy and architecture gaps; resolve dependencies or narrow scope. Reuse valid evidence; new APIs, major changes or changed contexts may need wider work. Learning and release readiness remain separate.

08 / THE CENTRAL INSIGHT

A clear answer starts behind the screen

An assistant can say “Done” with confidence only when the software behind it can establish what happened.



A useful capability needs API behaviour, application controls and people who sustain both. APIOps Cycles design-system scene, CC-BY-SA 4.0.

The editor and assistant need the same answer: is the approved article public, and where? A well-designed API helps both applications answer reliably. It also gives developers clear behaviour to build on. AI makes these existing dependencies more visible—and can spread their consequences across more actions and consumers.

> **Ask what “done” means**
>
> Choose one action your software carries out for someone. Can it establish the result, recover from uncertainty and explain what happened? Agree who keeps that experience reliable. That is a practical starting point for API quality.

The companion whitepaper, From API Readiness to Sustainable Quality, explores how APIOps Cycles connects assessment findings, business priorities and consumer needs with practical improvements that teams can sustain.

Continue the conversation

Find an APIOps Cycles partner

https://www.apiopscycles.com/partners/

09 / REFERENCES AND SCOPE

Publication details and sources

APIOps Cycles whitepaper series · Part 1: Understand the dependency.

Marjukka Niinioja · Osaango Ltd · 17 November 2026. Paper 1 in the APIOps Cycles whitepaper series. Companion: From API Readiness to Sustainable Quality (Paper 2). Method content and original visual artwork: CC-BY-SA 4.0; retain individual creator credits.

References

1. MCP · Tools
   https://modelcontextprotocol.io/specification/2026-07-28/server/tools
   Tools specification 2026-07-28; descriptions, schemas, results and access controls checked 16 November 2026.
2. Google AIP-155 · Request identification
   https://google.aip.dev/155
   Request identifiers and retry guarantees.
3. Google AIP-192 · Documentation
   https://google.aip.dev/192
   Clear descriptions for API consumers.
4. APIOps Cycles · Partners
   https://www.apiopscycles.com/partners/
   Find practitioners who can help apply the method.
5. APIOps Cycles · Explore the method
   https://www.apiopscycles.com/
   Connected business, architecture, design and operating-model perspectives; open method resources.

Background: Marjukka Niinioja, Beyond the API Economy: Productizing Digital Capabilities with APIOps Cycles (2026), pp. 20–21, 37–42 and 85–95. Supplied by the author; informs responsibility, capability and learning discussions.

The API designs, organisational histories and improvement trial are fictional. No task tests, savings or scores were measured. Technical references checked 16 November 2026. Visuals reuse the APIOps design system.