POSITION PAPER · TWO

The Deployment Lifecycle of AI Agents.

Why enterprise AI needs release management, not prompt management.

ENTERPRISE ARCHITECTURE

18-MINUTE READ

PAPER 02 · 2026

WHY THIS PAPER EXISTS

The first paper said *ship it*. This one is about what shipping means.

The companion paper argued that an enterprise AI agent is both a user and a program — it logs in like a person and **ships like software**. That was the easy half of the sentence. The hard half is the verb.

If an agent ships, it has a release process: it is drafted, evaluated, promoted, and — when it goes wrong — reversed. This paper is about that process, and about the quieter practice most teams are using in its place. Written for the architects and technology leaders who will own an autonomous principal in production, and who already know how much rides on how change reaches it.

Every Save button is a Release button.

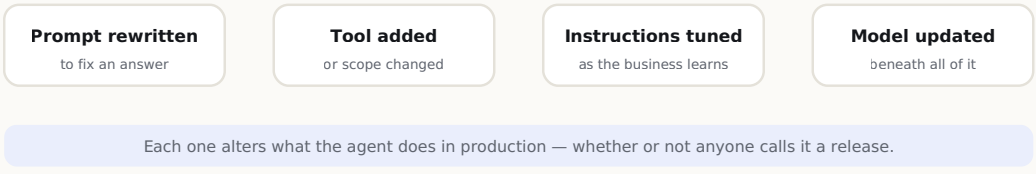
You just can't see the release. That invisible release — a change to an autonomous principal, reaching production with no version, no rehearsal, and no way back — is where enterprise AI risk actually lives. The rest of this paper is about making it visible.

THE PREMISE

An agent is never finished. It is only currently deployed.

A prompt is rewritten to fix a bad answer. A tool is added, or narrowed. Instructions are tuned as the business learns what it wants. The underlying model is updated beneath all of it. **Every one of these is a change to how an autonomous principal behaves in production.**

Static software is the exception that earned change control. An agent changes constantly — which makes change control not an optional maturity, but the central design problem.



Four routine acts. Four behaviour changes to a principal that acts on its own.

A change to production behaviour arrives through a text box and a save button.

We called it configuration.

It looks like editing a setting. It is closer to pushing code to production with no build, no review, and no version number. **The mistake isn't the prompt. The mistake is treating a deployment as a document edit.**

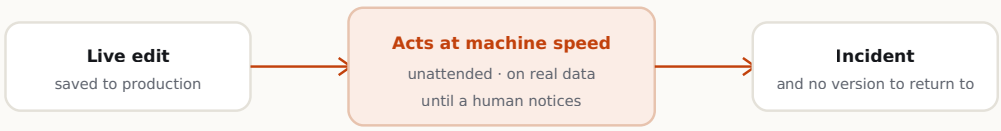
The interface says **configuration**. The consequence is a **deployment**. The label is where the discipline leaks out.

WHY LIVE EDITING FAILS

The first place an untested change runs is against real work.

Edit a prompt in production and three protections vanish at once. There is **no rehearsal** — the change meets real data on its first run. There is **no version** — you cannot say precisely what changed, or return to what worked. And there is **no containment** — an autonomous principal executes the new behaviour at machine speed, unattended, for as long as it takes someone to notice.

For static software that is a bad afternoon. For an agent that acts on its own, it is **unbounded operational exposure between the save and the next incident.**



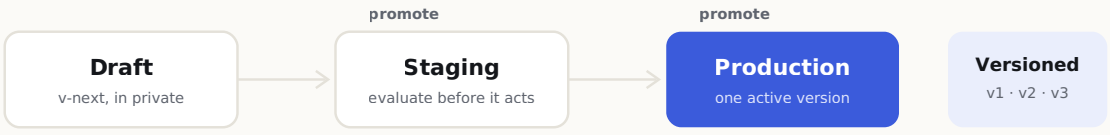
The window between the two arrows is the whole problem. A lifecycle exists to make that window impossible.

THE MODEL

A change earns its way to production. It does not arrive there.

The correction is not more caution. It is structure. A change begins as a **draft**, is evaluated in **staging** against representative work, and is **promoted** to production only once it holds up. Production runs exactly one active version at a time, and every version before it is kept.

Nothing here is novel. It is the shape of every mature software release, applied to a principal that happens to reason.



Promotion is a decision. Every prior version stays recoverable.

The companion paper named this lifecycle in one line. This is the line taken literally.

THE REFRAME

Rollback is not an engineering convenience. It is a governance control.

Rollback is usually filed under operations — a way for engineers to undo a mistake. For an autonomous agent it is something larger. It is the answer to the question every risk owner asks before approving autonomy: **when this goes wrong, how fast can we make it stop?**

With a lifecycle, the answer is one action: return to the last known-good version. Without one, the answer is an investigation. **The ability to reverse is what makes the freedom to act acceptable.**

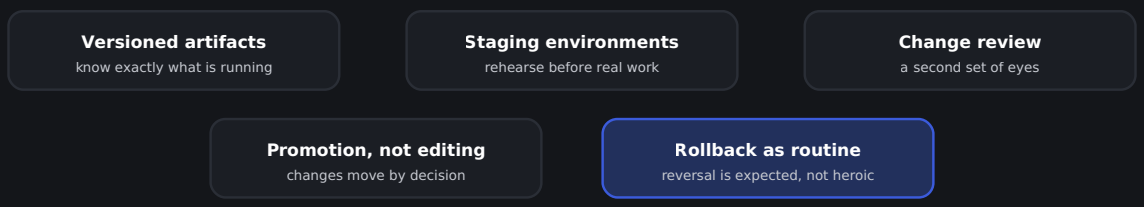
A risk owner does not need the incident to be rare. They need the reversal to be certain.

NOTHING TO INVENT

The discipline already exists. It just hasn't been pointed at agents.

Every practice in this paper was learned the expensive way, in software, decades ago. Versioned artifacts. Separate environments. Promotion instead of live edits. Change review. Rollback as a routine, not a rescue. **Enterprises already trust all of it — and audit against it.**

Treating an agent as something you release is not a new burden. It is the continuation of a discipline your organisation already runs, extended to its newest kind of participant.

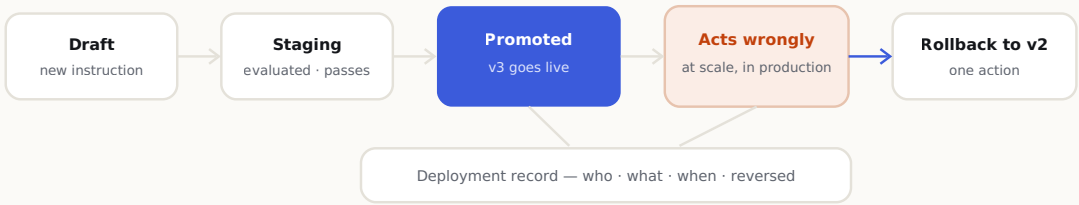


None of these were designed for AI. All of them were designed for exactly this problem: change reaching production safely.

PROOF — NO LONGER PHILOSOPHY

Watch a mistake reach production and get stopped.

A new instruction is drafted and evaluated in staging. It passes, and is promoted. In production it begins acting wrongly at scale — the kind of edge case no rehearsal caught. **Because it is a versioned release, not a live edit, the response is not a scramble.** The prior version is restored in one action, and the whole sequence — who changed what, when, and what it did before it was reversed — is already on record.



The change still failed. What differs is that failing was survivable, reversible, and — this is the point — **on the record**.

QUESTIONS ARCHITECTS SHOULD ASK

Ask how a change reaches your agent. Notice what kind of question it is.

Not one of these is a question about AI. **They are all release-management questions** — the same ones you already ask of anything else that ships to production.

01 **What version is in production now?**

02 **Where was this change rehearsed?**

03 **Can we say exactly what changed?**

04 **Can we return to what worked?**

05 **Who approved the promotion?**

06 **How fast can we roll it back?**

07 **Which version was running when it failed?**

08 **Is the change on the record?**

If any answer is "we just edit the prompt," the agent is running in production without a release process — and so is the risk.

There is no such thing as prompt editing. There are only unmanaged deployments.

An agent you cannot version, stage, or roll back is not deployed — it is merely running. Prompt management asks how to change an agent quickly; release management asks how to change it safely. When that distinction turns up in an architecture review that has never heard of the company that wrote it, the argument has done its job.

THIS IS NOT A PROPOSAL

This lifecycle already exists.

Most organisations will reach a release process for agents after an incident makes the case. Copyl was built around this lifecycle from the start.

ARCHITECTURE IMPLEMENTED TODAY

- **Versioned agent profiles**
- **Draft → Staging → Production**
- **Explicit promotion**
- **Single active production version**
- **Rollback to any previous version**
- **Execution, decision and activity logging**

copyl.com

ABOUT THE AUTHOR

Rolf Bäck

Founder & Chief Architect, Copyl

Rolf has spent more than 30 years building enterprise software platforms. Since 2011 he has led the development of Copyl, an enterprise platform for business applications, automation and AI. His current work focuses on how AI agents become governed participants in enterprise systems rather than isolated assistants.

- Founder of Copyl
- Enterprise software architect
- Working on enterprise AI architecture since 2022
- Based in Sweden

ABOUT THIS SERIES

This is the second paper in a series on the architecture of enterprise AI. The first, *AI Agents Are Users, Not Features*, argued that an agent is both a user and a program. This paper takes up the second half of that claim — what it means for an agent to ship — and the ones that follow take up identity, the second population of enterprise systems, and the operating system they will need.

copyl.com