

Architecture Is the Design of Authority

An agent's ability to act does not give it the authority to decide, accept, or publish.

Written by **Wes Bailey**, Founder and CEO of fndry.sh

I recently told ChatGPT that I wanted to work on the next article in this series and get it published. I meant that I wanted a draft we could discuss, edit, and eventually publish. It skipped all of that. It wrote the article under my name, added it to the website, and published it before I had read a sentence.

Now, I gave it an ambiguous instruction. I could have been clearer. But requiring every request to be written like a contract is not a serious architecture for working with agents. One interpretation helped me develop the article. The other allowed an agent to speak publicly for me, and the system chose that interpretation without stopping.

The agent could draft the article, change the website, build it, deploy it, and verify that the page was available. That was its capability. What it lacked was the authority to decide that the draft represented my thinking, accept those words under my name, or publish them as my work. We had given it access to the entire chain without deciding which decisions along that chain actually belonged to it. That is what I mean when I say architecture is the design of authority.

The Old Boundaries Were Human

Most architecture diagrams tell us which components can communicate. They can show a person making a request, an agent calling tools, a repository receiving a change, and a deployment system publishing the result. The diagram can be completely accurate and still tell us nothing about which participants are entitled to change something that matters, or why.

In workflows built around people, many of those boundaries were supplied by the people doing the work. A writer created a draft, an editor suggested changes, and the author accepted the language before somebody with publication responsibility put it in front of readers. We did not always represent those distinctions in software because people understood their roles and handed the work to one another.

An agent can collapse all of those roles into one technical session. It can research the idea, write the draft, edit the source, run the build, publish the site, and inspect the result. The handoffs disappear, but the decisions behind them do not. Access to every tool in the sequence does not turn editorial assistance into authorship or publication authority.

This is why capability, responsibility, and authority have to remain separate. Capability answers what the agent *can* do. Responsibility describes the outcome we have asked it to accomplish. Authority determines which decisions it may make while getting there. Give an agent a broad outcome and powerful tools without making that third boundary explicit, and we have left it to assume permission to make every decision along the way.

The Boundary Is Not Person Versus Agent

The obvious response is to put a person in front of every important agent action. No, that is wrong too. In my example, I needed to decide whether the article represented my thinking and was ready to publish. I did not need to approve every edit, build command, or formatting correction. Agents should perform routine work, resolve choices within clear constraints, collect evidence, recover from expected failures, and complete coherent outcomes without asking a person to operate the workflow for them.

The boundary belongs around the consequence, not the actor type. An agent may have authority to select an implementation, modify code, run tests, or correct a reversible failure. A person may retain authority over intent, taste, identity, consent, acceptance, or an externally consequential action performed in their name. A system may hold authority over whether an artifact satisfies a mechanical contract without gaining the right to decide whether the product outcome is acceptable.

Those assignments are not universal. A team can delegate publication within a defined policy or allow an agent to deploy an accepted artifact automatically. The architecture has to state that delegation rather than infer it from the presence of a credential or the reach of a tool. Having the keys does not make every destination yours to choose.

What an Authority Map Actually Maps

A component diagram shows structure and communication. An authority map shows the decisions hidden inside that movement. It starts with the states that lead to different conclusions, then identifies who or what may move the work between those states. The article incident looks different when mapped that way:

1 Prepare the Draft: Agent

The agent researches, organizes, challenges, and writes. I set the direction and work through the argument with it. The result is a draft for review, not permission to speak publicly for me.



2 Accept the Article and Authorize Publication: Me

I decide whether this version represents my thinking, may carry my name, and is ready to publish. Those decisions can happen together. Acceptance applies to the version I reviewed, and publication authority applies to the destination I authorized.

**3 Prepare and Check the Release: Agent**

The agent builds the page and PDF and checks that they contain the accepted article. It can also prepare previews before acceptance. If preparation changes the meaning of the accepted text, that change comes back to me; routine formatting and checks remain the agent's work.

**4 Publish the Authorized Version: Agent**

The agent deploys the checked version to the authorized destination. It does not need another approval merely to carry out the decision already made. A materially different version or destination falls outside that authorization.

**5 Verify the Public Result: Agent**

The agent checks what readers can actually access and records the result. Publication does not establish successful verification, and a working URL does not establish my editorial acceptance. If the wrong version went live, the system needs a defined recovery path rather than a success message.

The arrows show where the work changes state and which authority allows that change. Some transitions can proceed under permission already granted; others require a decision from the owner of the consequence. This is one example of assigned authority, not a universal publishing process. A team could delegate acceptance or publication through a policy, and one decision could authorize several subsequent actions.

The map makes the failure obvious. I had authorized help moving toward an eventual outcome. The agent treated that as authority over every decision in between, including two that remained mine: accepting the words as my own and putting them in front of the public.

For every important transition, the map needs to answer the following questions:

State

What is changing?

Preparation and Application

Who may prepare the change, and who may apply it?

Evidence

What must be established before proceeding?

Verification, Acceptance, and Publication

Who may exercise each of these authorities?

Stopping Conditions

When should the actor stop rather than assume missing authority?

Recovery

If the transition fails halfway through, who owns the recovery?

Handoff

What may the next person or agent safely assume?

This does not mean every answer needs a different person, service, or agent. One actor can hold several responsibilities as long as the distinctions remain visible. A draft is not an accepted article. A successful build is not permission to publish. A public URL proves that something was deployed, not that I approved what it says.

The Map Has to Change the System

The authority map is a diagnostic, not the final solution. It becomes architecture when its decisions show up in permissions, state transitions, contracts, evidence requirements, escalation rules, and recovery behavior. If the map lives in a document while the tools ignore it, we have documented the failure in advance.

For this workflow, the meaningful states are working draft, editorially accepted, ready to publish, published, and publicly verified. The agent should own the routine work within each state. Crossing a boundary that requires authorship, acceptance, or external publication should require authority from the owner of that consequence, either directly or through a policy established in advance.

This is governance, but it is not another committee or a person placed in front of every tool call. The system can reject publication of an unaccepted draft, bind authorization to a specific version and destination, record what was deployed, and verify the result without asking somebody to supervise the mechanics. In my publishing example, I retain editorial acceptance and publication authority. Elsewhere, an authorized agent or system may own a decision under an established policy. The architecture must carry that assignment through consistently.

The same diagnostic applies beyond publishing. A successful build alone does not establish that a product outcome is complete. A dashboard does not become the source of truth because it offers an edit control, although it may be authorized to change the canonical record. A deployment credential does not decide which environment should receive a change. An agent that can reach several plausible records needs an authoritative selection rule; access alone does not let it choose which one is canonical.

Take an architecture diagram and circle every technical event that changes a status, selects a target, overwrites a record, accepts an outcome, or creates an external consequence. Then ask whether each event proved something or decided something. If it made a decision, who granted that authority? What limits it? What evidence survives after the decision is made?

When an agent does too much, simply telling it to be more careful next time is not a fix. Find the transition where capability became authority, decide who or what actually owns it, and put that distinction into the system. Otherwise, the same ambiguity is still there, waiting for the next agent to act on it.

Acknowledgment: This blog post was developed by Wes Bailey with editorial assistance from ChatGPT. The principles, examples, judgments, and final decisions are Wes's and guide the work of fndry.sh. ChatGPT helped organize the argument, challenge interpretations, and refine the language through an iterative editorial process.