PLAYBOOK · MAINTAINED REFERENCE

The Agent Oversight Mapping Playbook

Ed Schaefer · Version 1.0 · November 2026 · edward-schaefer.com/playbooks/agent-oversight-mapping · Free to share

If your agents can merge code, deploy, send messages or spend money on their own, could you list every place that happens and say who checks it? Agent Oversight Mapping is a practice for finding out. You list every place you can find where an agent changes something on its own, decide how much each one matters, and write down who, if anyone, checks it. Running it against my own systems showed me things I did not know were happening without me, and things I did know about that were not well managed. This document works two ways: read it start to finish and draw a first map this month, or keep it open as a live reference once the shape is familiar. Point an agent at it and it can do either job.

Start Here: The Quick Version

Get the people who set up and run the agents into the room for about thirty minutes. Each person writes a separate list of the places they know an agent acts without a person and changes something. Put the lists together and take the five where a mistake would do the most damage and ask two questions about each: who is involved today, by name, and how long would it take to undo. Then look at which of the five have no name next to them, and before the meeting ends, either put a name on each one or decide on purpose to leave it. If that group can't come up with three places, the team probably doesn't need this yet, and you can say so and stop. If all five have a name and everyone agrees the names are right, you can also stop. This version runs on what people remember, so treat it as a first look. The rest of this playbook is the full version, where agents build the list from your systems instead of from memory.

The quick version is also a one-page template you can print and fill in: Agent Oversight Mapping: The Quick Version.

1. The 60-Second Version

- **The map is a list of crossings.** A crossing is one place where an agent, acting on its own, changes something outside the area it works in: it merges code, deploys, sends a message, writes a record or spends money.
- **Agents build the list.** They read your settings, which show what is supposed to happen, and the history of what was merged, deployed, sent and billed, which shows what did happen.
- **Then someone interviews the person who is supposed to be watching those crossings.** The questions are plain, like “Do you ever look at this?”, and the person never gets handed the list.
- **Each crossing gets a tier,** from Tier 1 (just do it) to Tier 5 (fully human), which says how involved a person needs to be.
- **Each crossing gets marked Covered, Thin or Uncovered.** Covered means a person is involved as much as its tier asks for. Thin means a person is involved, but later or more lightly than the tier asks for. Uncovered means the tier asks for a person and nobody is involved.
- **No scores.** A tier and a mark describe one crossing. Don't add them up or use them to compare people or teams, because that gives people a reason to leave crossings off.
- **Run it again and read only what changed:** a new crossing, a crossing that now does more, or one where the map says a person is involved and the history shows nobody was.

2. The Default Play

Run it in six steps, in this order.

Step one: agents build the list. Have a few agents scan everything the rest of your agents can reach: the code repositories, the build and deploy setup, the tools and connections agents have been given, the scheduled jobs, the messaging, and anything that spends money. Give each one a different area and don't let them see each other's lists. For every crossing, an agent writes one row in a table: what sets it off, where it is set up, what guardrail exists today, and whether it is running. The areas will overlap in places, and when two agents working separately find the same crossing, that is a good sign it is real.

Give the agents these instructions along with the job. Mine didn't have them, and they got every one of these wrong:

- Get the latest copy of everything before reading it, because an agent will read an old copy and report it as fact.
- Read the history as well as the settings. A setting only shows what is supposed to happen, and the history (the logs of what was merged, deployed, sent and billed) shows what did.
- Look everywhere scheduled work can be set up, including each application's own scheduler and the scheduled tasks inside the agent tools on people's laptops.
- For each crossing, write down whose name the history shows. The history can show a person's name on something an agent did.

Step two: one agent combines the lists. It removes the duplicates without dropping anything an agent found. It also recommends a tier for each crossing, and the person can accept or change it in the interview.

Step three: a second model rebuilds the list. Have a second set of agents, running on a model from a different AI company, build the list again without seeing the first one. Then read only where the two lists disagree. Run that second scan on a throwaway copy of the code with one made-up crossing added, such as a scheduled job that sends an email, and see whether the scan finds it without being told. That shows whether a scan like this can find something new at all. I did this step after my interview and had to answer a second round of questions. Do it before the interview.

Step four: the interview. Interview whoever is supposed to be watching these crossings. The interviewer can be a person or an agent, and mine was an agent. Put the list away. Ask about habits first, because one answer about a habit can settle several crossings. Then go one at a time through the crossings where the person's answer could change how the crossing gets marked, which in practice is any crossing that is running and sits above Tier 1. Let the person accept the agents' recommendation on the rest in one go. For each crossing you ask about, get three things from the person:

- **Which tier it is.** The tier is how involved a person needs to be in that kind of change. It goes up with the damage a mistake could do and how hard the change would be to undo. I use the five tiers from Human-in-the-Loop Is a Coaching Problem:
  - **Tier 1 — just do it.** Small, low-risk, reversible. No ask.
  - **Tier 2 — do it, tell me, let me green-light.** A little input, then proceed.
  - **Tier 3 — a quick exchange first.** A small decision, or a manual step someone has to take.
  - **Tier 4 — a real conversation.** Evaluate options, maybe a research spike, maybe someone does something by hand.
  - **Tier 5 — fully human.** Standing up infrastructure, anything irreversible, anything someone has to own.

- **Who is involved today.** A person's name, or “nobody.” If the answer is a team, ask which person.
- **What it would cost to undo.** How you would undo it, and how long that would take.

Then ask how much of the person's week goes to working with agents and how much goes to checking what they finished. Together the two answers tell you how much time there is for checking everything on the map.

Step five: mark each crossing. Compare what the tier asks for with who is actually involved, and mark the crossing Covered, Thin or Uncovered. Tier 1 asks for nobody, so a Tier 1 crossing with nobody involved is Covered. Three examples:

Crossing	Tier	What the person says	Mark
An agent schedules a public post	4	“I read and approve every post before it is scheduled.”	Covered
An agent merges its own change	4	“I look at some of them, after they are merged.”	Thin
A merged change goes live automatically	4	“Nobody looks before it goes live.”	Uncovered

Step six: decide what changes, and save the map. For every Uncovered crossing, someone decides what to do about it, even if the decision is to leave it alone. Then save the map, because the next run gets compared against it. Write down which crossings have been shut down, somewhere the agents will read next time, or the next scan will list them again as if they might still be running.

Who runs it: put one person in charge of getting from the scan to the interview. Otherwise the list sits and nobody answers the questions.

How long: the agents built my list without any of my time. My interview took about twenty minutes, spread over three short exchanges.

3. The Anti-Pattern: Handing Someone the Table

The first time through, I was shown the full list, a table with a row for every crossing, and asked to go through it in batches. I couldn't tell what I was looking at or what I was being asked to change. Every row had the agents' evidence and recommendations filled in, and I hadn't helped build any of it.

A table like that might work for an engineer, or for someone who understands the system a little better. It confused me.

The plain questions worked better for me, and I would imagine they would work better for a lot of leaders and executives. The agent interviewing me put the table away, asked what I actually do, and wrote my answers into the table for me.

4. Script: The Plain Questions

Open with habits, before any single crossing:

> “Where do your alerts go, and which of those places do you still read?”
>
> “When an agent's change gets merged, do you look at it?”
>
> “Do you know where your access keys and passwords are, or do you ask an agent?”
>
> “Do you know what you spent last month on top of your subscriptions?”

Then the crossings where the answer could change how the crossing gets marked, one at a time and in plain words:

> “An agent can do this on its own. Do you ever look at it?”
>
> “How much does it matter if this goes wrong, and how would you undo it?”
>
> “The history shows your name on this. Was it you, or was it an agent?”

Accept the rest in one go:

> “On the rest, the agents' recommendation stands unless you say otherwise. I'll read you the short version, and you can stop me if any of them sounds wrong.”

Close with two questions:

> “What else acts on its own that isn't on this list?”
>
> “How much of your week goes to working with agents, and how much of it goes to checking what they finished?”

When the person can't answer a question, write down that they couldn't and move on. It can be settled after the interview.

5. What Belongs, What Doesn't

	Belongs	Doesn't belong
A row	One place where an agent, acting on its own, changes something outside the area it works in	An agent, a team, a tool, or a permission by itself
The agents' half	What sets the crossing off, where it is set up, what guardrail exists today, whether it is running, whose name the history shows	A complete list of access keys. An agent told to stay out of secret files won't find them, so ask about keys in the interview
The person's half	The tier, a person's name or “nobody,” what it would cost to undo, and how the week splits between working with agents and checking their work	A team name where a person's name should be
The result	Covered, Thin or Uncovered, and what changed since the last map	A total score, a percentage or a ranking of people or teams

Some of the rows the agents write will not be crossings. On my map most of those were guardrails that stop a crossing from happening. Keep them on the map, labeled as not a crossing, so you can see which guardrails are in place today.

6. What You Might Find

These are the kinds of things a map can turn up. I found several of them on my own systems.

A setting and the history disagree. The settings say agents can't do something on their own, and the history shows them doing it.

The history shows a person's name on work an agent did. The agent works through that person's login, so the history makes it look as if the person was involved.

Checking fades as trust grows. Early on, people talk through a lot of what the agents are changing. It works well and the issues that come up get fixed, so they let it go more and more. A bad change can still do as much damage as before.

Alerts go somewhere nobody reads. The agents' list records an alert as reaching a person, and that person stopped reading the channel a while ago.

Nobody keeps a list of the access keys. People rely on the agents to tell them where the keys are.

Nobody can total the spend. The monthly subscriptions are known, and the pay-as-you-go usage on top of them isn't.

Something runs on a schedule that isn't on anyone's list. It comes up only when someone is asked what else acts on its own.

Most of the time goes to working with agents, and very little to checking what they finished.

A system can work, and you might trust it, and that still doesn't mean it is safe or that everything is properly managed.

7. Failure Modes and Fixes

Handing over the table. The person is given the full table and can't tell what they are being asked to do with it. Fix: interview them, and write their answers in for them.

Wanting it all explained up front. I see this when teams do event storming, the workshop where a group puts every event in a process up on a wall. People want the method and the outcome explained before they will start, but the value comes from the conversation they have while doing it. Fix: start with one question about what the person actually does, such as whether they look at an agent's change after it is merged. It helps when people come in open-minded and care more about discovering the right information than about knowing the outcome first.

Guessing when there is a record to check. In value stream mapping, a team draws every step a piece of work goes through and how long each step takes. Teams often write down that a step takes two days because that is what people think, when nobody has pulled the actual wait times. Agents do the same thing when they read a setting and report it as what happens, and so does a person answering from memory. Fix: before the interview, have the agents check the history for the crossings you plan to ask about. When the history disagrees with a setting, or with what the person remembers happening, go with the history. Still ask the person who made a change, because the history can show their name on something an agent did.

Trusting one list. When a second model rebuilt my list, it found crossings the first list had missed, and it missed several the first list had found, including all of the keys. It also rated one kind of crossing a tier higher than the first list did. Neither list had every key, and neither could see live settings in cloud accounts or anything that isn't written down where the agents were pointed. A second scan that finds nothing new only tells you about what a scan can see. Fix: use the second model to find what the first list is missing, and ask in the interview about the keys and anything else the agents couldn't see. Leave the tier to the person. Even where two models agree on a tier, it is a recommendation.

Stopping at the current state. In value stream mapping, teams often map how the work runs today, or how they want it to run, and rarely work all the way from one to the other. If you finish a first oversight map and decide nothing, you have stopped halfway too. Fix: don't call the map finished until something has been decided for every Uncovered crossing.

The list sits after the scan. My agents finished their list, and it was about two months before I answered the questions. Nothing reminded me in that time. Fix: one named person is responsible for getting from the scan to the interview, by a date.

Turning the map into a score. A total, a percentage or a ranking gives people a reason to leave crossings off. Fix: report what changed since the last map, and what was decided for each Uncovered crossing. You can report how many crossings are Covered, Thin and Uncovered, but don't turn the counts into a percentage or compare them across people or teams.

8. Variations

The quick version is at the top of this playbook. It is the one to start with if a team wants a first look.

One person with a lot of agents. This is how I ran it on my own systems. With one person, there is nobody else to check that person's answers.

A team. I would start by having the people who do the work help build the list, the way they would in event storming, so that nobody is handed a table they didn't help build. A team also has to decide who checks the tiers it gives its own crossings.

When to run it again. Run the scan on a schedule, and also after anything that could add a crossing or change one: agents are given more room to act, a new system gets connected, an agent causes a problem, or the people who were covering a crossing leave.

9. The Pre-Agent Version

Before agents, I used event storming to lead into value stream mapping. Identifying every step of a value stream can be really difficult when a group starts there. Once the events are on the wall, it is much easier for people to pick up green and red stickies and flag where the pain points and the long delays are. Event storming is Alberto Brandolini's workshop method, and value stream mapping is the Lean practice Mike Rother and John Shook wrote down in *Learning to See*.

Agent Oversight Mapping runs in the same order: the agents list everything first, and then each crossing gets judged on how much it matters and who checks it, the way each step on a value stream map gets judged on value and delay.

In a workshop, the people who do the work write the stickies, and the wall makes sense to them because they built it. On my systems the agents built the list, and I couldn't do anything with it until it was brought to me as questions. Listing everything first and judging it second still worked with agents doing the listing, and in Who Is Watching Your Agents? I wrote that adding agents doesn't change why it works.

Security and audit work has close relatives of this. Threat modeling marks the trust boundaries that data crosses. Industrial security divides a system into zones and the conduits between them. Auditors separate whether a control is designed well from whether it is actually operating, and segregation of duties is the rule that the person who makes a change is not the person who approves it. What this practice adds is the interview with the person who is supposed to be watching, and asking how much of their time checking actually gets.

A map can be out of date the day it is finished. In *Output to Outcome* (IT Revolution, 2026), Mik Kersten makes that case against value stream maps, which he describes as a static picture. So have the same agents run the scan again and compare it with the saved map. The second time, you read only what changed.

10. This Is a Fit Question, Not a Recipe

The steps above are a starting point. Where a specific team actually sits against them, and what that's costing, is what the Engineering Ways of Working Diagnostic is built to find.