Evals are the missing layer in agentic software

Agent demos are fun, but shipping agents is a different sport. The moment an agent is allowed to take actions, you stop debugging a single output and start debugging a trajectory.

Anthropic's *Demystifying evals for AI agents* is useful because it treats evaluation as engineering infrastructure, not as a "research benchmark." The claim is not that evals are a nice-to-have; it's that without evals teams get trapped in reactive loops: fix one failure in production, accidentally create three new ones, and never know whether the agent is getting better or just getting different.

What an eval is (and what changes with agents)

At the simplest level, an eval is: give an AI an input, then apply grading logic to measure success. In a single-turn setting, that's already familiar: a prompt, a response, and a score.

Agents complicate this because they operate over multiple turns, call tools, and modify state. That means success is often not "did the assistant say the right thing?" but "did the environment end up in the right state?"—e.g., did the reservation actually get created in a database, did the code compile and pass tests, did the ticket really get resolved.

This framing—grading the *outcome* and sometimes also the *transcript*—is the first big conceptual shift. If the agent is doing work inside a sandbox, the only reliable truth is what changed in that sandbox.

Vocabulary that matters in practice

I like when a post insists on definitions, because it forces clean mental models. In Anthropic's taxonomy:

- A **task** (test case) is one problem with inputs and success criteria.
- A **trial** is one attempt at a task (you run multiple because outputs vary).
- A **grader** is the scoring logic (and a task can have multiple graders).
- A **transcript** is the full trace: messages, tool calls, intermediate results.
- The **outcome** is the final environment state (often more important than the transcript).
- An **evaluation harness** runs tasks end-to-end and aggregates results.
- An **agent harness** (scaffold) is the orchestration layer that makes the model an "agent" (tool selection, loops, stopping conditions).

The subtle point is that when you "evaluate an agent," you are evaluating the harness + model as a coupled system. That's closer to how software is actually shipped.

Why evals compound (and why teams delay them)

The post describes a common lifecycle: early on, teams can get far with manual testing, dogfooding, and intuition. It feels fast and it feels like progress.

The breaking point arrives later: users complain the agent got worse after a change, and the team is flying blind. Without evals, debugging becomes an incident response loop. With evals, failures become tasks, tasks become regression tests, and the whole system becomes legible.

Another pragmatic benefit is model upgrades. If a team has an eval suite, swapping the underlying model becomes less like a leap of faith and more like running a test battery, then tuning prompts/harness until the metrics recover.

Three grader types (and why you need all three)

Anthropic emphasizes mixing graders because each has different failure modes:

- **Code-based graders**: unit tests, exact checks, static analysis, state checks. Fast, cheap, reproducible, but brittle and sometimes blind to nuance.
- **Model-based graders**: rubric scoring, comparisons, natural-language assertions. Flexible and scalable for open-ended behavior, but non-deterministic and requires calibration.
- **Human graders**: expensive and slow, but the gold standard and the calibration target for the other two.

This is one of those "obvious in hindsight" points: if an agent is doing something that has crisp ground truth, use deterministic graders; if it's doing something inherently subjective (tone, helpfulness, over-engineering), you need rubrics and humans.

Capability evals vs regression evals

A distinction worth stealing for any engineering org:

- **Capability** evals ask: "what can this agent do?" and should start with a low pass rate. They define a hill to climb.
- **Regression** evals ask: "did we break anything?" and should aim for near-100% pass rate.

As capability improves, some tasks "graduate" into regression suites. This is basically continuous integration, but for agent behavior instead of just code.

Non-determinism: pass@k vs pass^k

Agent systems are stochastic, so a single run is not a measurement. Anthropic highlights two metrics with different product meanings:

- **pass@k**: probability of at least one success in k tries (good when "one working solution" is enough).
- **pass^k**: probability of succeeding on all k trials (good when users expect consistent success every time).

This is a clean way to express a product requirement as a metric. Some agents are allowed to be "creative" across attempts; others must be boringly reliable.

A practical roadmap to start

The roadmap in the post is refreshingly 80/20. The core ideas:

- Start early, even with 20–50 tasks drawn from real failures.

- Write unambiguous tasks; if two domain experts wouldn’t agree on pass/fail, the task is broken.
- Create a reference solution per task to ensure the graders are configured correctly.
- Build balanced problem sets (test both when a behavior should happen and when it shouldn’t).
- Stabilize environments so trials start clean and noise comes from the agent, not the harness.

The meta-lesson is that evaluation design is part of product design: you are deciding what “good” means, then making it executable.

Contributor: Alessandro Linzi