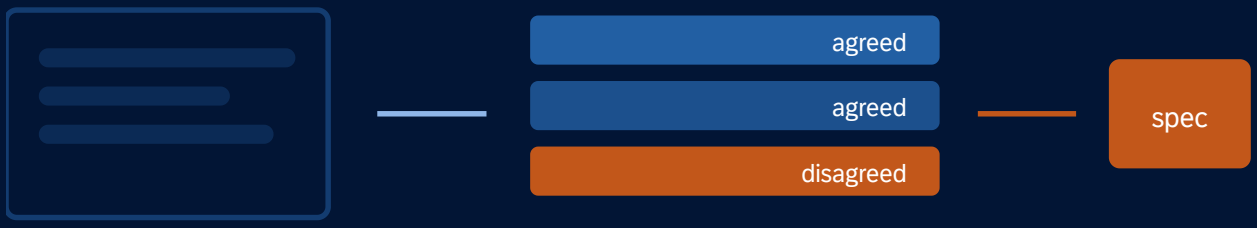
WHITEPAPER · AGENTIC AI FOR PRODUCT LEADERS

Prototyping the Judgment

What a product manager builds when the screen is no longer the product



Product managers are now expected to prototype. At high autonomy there is almost no interface left to prototype, which looks like a contradiction and is not. The object of the prototype moved. What you build, who validates it, and why the disagreements are the specification.

Yoram Friedman, MD
Physician and enterprise product leader · SAP, Walmart, Microsoft, regulated health technology
Drawn from the Agentic AI for Product Leaders series · agenticaiproductmanagement.com

THE ARGUMENT

Watch the prototypes people are sharing. Almost every one of them is a deterministic UI.

Dashboards. Forms. Onboarding flows. Landing pages. CRUD screens with a sidebar. The demos circulating to prove that product managers can now build are, with very few exceptions, demos of software where the screen is the product.

That is exactly the case where the technique works, and it is why the technique looks universal. Nobody is generalizing carelessly. They are generalizing from the only examples anyone shows.

Two expectations are now placed on the same person, and in the agentic case they appear to contradict each other. Prototype the thing, validate it with customers, and demonstrate it to engineering, without waiting for design or development. At the same time, build products where the agent acts and the human interface shrinks to a conversation, an approval, and an occasional escalation.

So the question a working PM actually asks is fair, and I have never seen it answered well. **If the user is no longer in the room, what am I supposed to prototype, who tries it, and how does any of it help engineering write boundaries and policies?**

> The contradiction comes from an assumption nobody states out loud: that a prototype's job is to show an interface. It never was. The screen was the cheapest available proxy for the behavior, and when behavior was deterministic those were the same act. For an agent they come apart.

Which means prototyping does not stop at high autonomy. Its object moves, from the interface to the judgment.

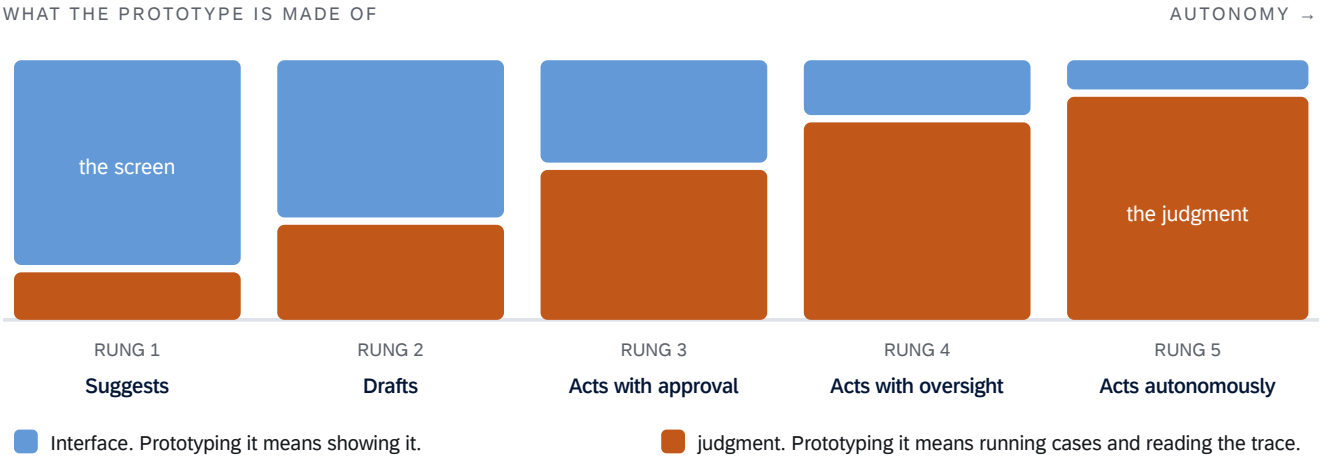


Figure 1. What a prototype is made of, by rung. At the bottom of the ladder the screen is most of the product and prototyping it means showing it. At the top there is barely a screen, and what needs validating is whether the system decides the way a competent person would. Schematic, not measured.

PART ONE

What you actually build

Not a clickable artifact. A throwaway agent wired to twenty real cases, whose output is a decision log rather than a screen.

You are not building something to be experienced. You are building something to be run. Twenty cases in, twenty decisions out, each one carrying what it decided, what it relied on, and where it was uncertain.

This is what vibe coding is unusually good at and is currently under-used for. Production quality is irrelevant. Error handling is irrelevant. Nobody will ever click it. It has to load a case, reason, act inside whatever bounds you have guessed at so far, and print a trace you can read over someone's shoulder.

The build, concretely

The cases. Twenty real ones, pulled from the system of record rather than invented. Weighted deliberately toward the hard end: the ambiguous, the just-outside-policy, the one that looks routine and is not. The easy cases teach you nothing, because the easy cases never needed a person.

The agent. Whatever runs. A prompt, a couple of tools, a loop. An afternoon.

The output. One row per case: the decision, the reasoning, the confidence, and whether it would have escalated. Plain text is fine. A spreadsheet is better than an interface.

What you do not build. A UI. Not even a small one. The moment you build one, the conversation in the room becomes about the UI, and the UI is not the thing in question.

Notice that this is cheaper than what most teams currently do at this stage, not more expensive. It is a smaller artifact answering a bigger question.

PART TWO

Who validates it, since the user has left the room

Two people can, and neither is the persona usability testing was built for. A third cannot, and that is the part worth sitting with.

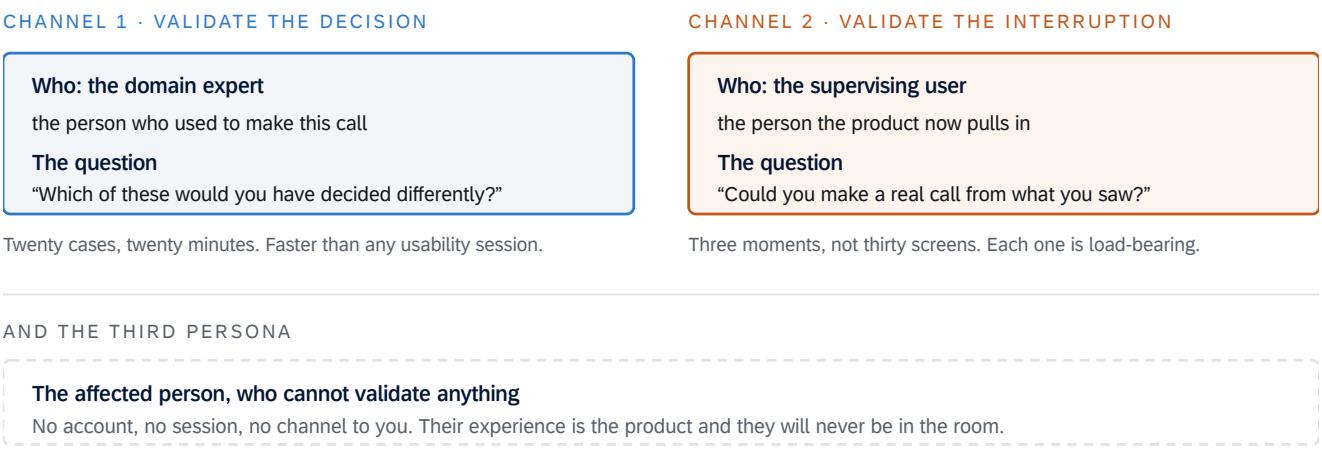


Figure 2. Two validation tracks, one per channel, plus the persona who has no way to participate in either. The tracks answer different questions and should not be run in the same session.

The domain expert validates the decision

The person who used to make this call. Not a proxy, not a stakeholder, the practitioner. Put the twenty rows in front of them and ask one question: which of these would you have decided differently?

That is a session rather than a study, and it is more decisive than any usability test you have ever run, because the expert is not being asked to imagine using something. They are being asked to adjudicate work in their own domain, which they do every day and are fast at. Most of the twenty go quickly. The handful that do not are the session, and they are the reason you built the thing.

It also fails safe in a way usability testing does not. A user who cannot find a button tells you the button is wrong. An expert who disagrees with a decision tells you the product is wrong, immediately, in a way that cannot be argued into a backlog item.

The supervising user validates the interruption

The person the product now pulls in when the agent hands over. The surface they see is thin, a conversation, an approval, an escalation, and thin does not mean unimportant. At rung five you are prototyping three moments instead of thirty screens, and every one of the three is load-bearing in a way no dashboard screen ever was. Value per pixel goes up as the interface shrinks, not down.

The question here is not whether the interface is pleasant. It is whether a competent person could make a real call from what they were shown, in the time the workflow actually gives them. A prototype of an approval moment that takes ninety seconds to read, in a queue that allows fifteen, has failed and you can know that before you build it.

You do not have to invent what goes into those three moments either. It comes out of the first track: whatever the domain expert asked to see before they would commit to a decision is, almost exactly, what the supervising user will need to see. Part three turns that into an artifact.

The affected person, who cannot validate anything

The third persona is the one the decision lands on. No account, no session, no channel to you, frequently not aware there was an agent. They cannot be recruited for a session because they are not users.

The nearest available substitute is not research, it is representation: someone in the room whose explicit job during the review is to speak for them, and a set of cases in the twenty

chosen specifically because that person is the one who gets hurt if the agent is wrong. It is a weak instrument. It is better than the alternative, which is discovering the gap through a complaint.

PART THREE

The disagreements are the specification

This is the part that resolves the tension with engineering, and it is a better handoff than a clickable prototype ever produced.

A screen prototype gave engineering a target to implement. A judgment prototype gives them something more useful: the places where a competent human and the system parted company, observed rather than imagined.

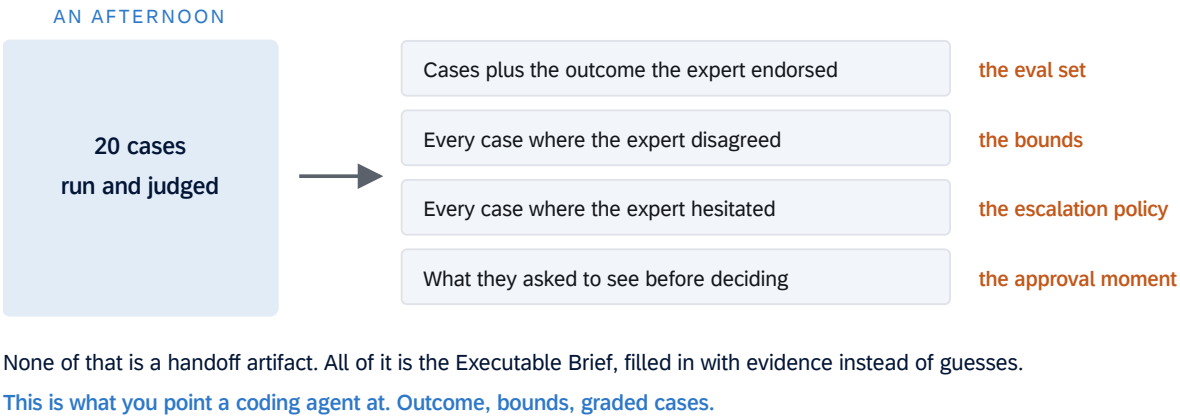


Figure 3. What falls out of one afternoon. Four artifacts, none of which is a handoff document, all of which go directly into the executable brief.

The cases the expert endorsed are not feedback, they are the golden dataset the eval suite will later grade against, assembled by the one person qualified to say what an acceptable answer to an ambiguous case looks like.

Every case where the expert said *no, I would have escalated that* is a boundary condition, discovered empirically. You cannot write *escalate on ambiguous fraud signals* from an armchair with any useful precision. You can find exactly where that line sits in twenty cases in an afternoon, and you can find it with the person whose judgment the line is supposed to encode.

Every case where they hesitated before answering is an escalation candidate, and hesitation is more informative than disagreement, because it marks the cases where the right answer is

contested rather than merely missed.

Every case where they asked to see something before deciding is the content of the approval moment, specified by the person who will need it rather than guessed at by the person building it.

> Vibe coding did not become less useful at high autonomy. It changed what it produces. It used to produce a demo. Now it produces evidence.

And this is precisely what a coding agent needs pointed at it. Not a design file. An outcome stated as a reasonable senior operator would state it, the bounds around it, and a graded set of real cases each paired with the call an experienced person actually endorsed. Engineering can build to that. Engineering cannot build to a screen that describes a person who will not be there.

PART FOUR

The trap, and it is a serious one

A working agent is the most dangerous artifact in this entire discussion, and it is dangerous in a way a screen mockup never was.

The moment a prototype demos well, the go or no-go decision leaves the room and does not come back. Nobody kills a thing that works. That has always been true of prototypes, and at high autonomy it gets worse, because the agent does not need a polished interface to be convincing. It just has to answer well in the meeting.

So the discipline is the one the practice was named for and then quietly abandoned: **prototype to decide, never to show.**

Which has a practical consequence for the review. What you put in front of the room is not the running agent. It is the twenty rows and the column showing where the expert disagreed. If someone asks to see it run, that is the moment to notice that the conversation has moved from whether this should exist to how impressive it is.

The rule that keeps it honest

A prototype that cannot produce a no is not a prototype. It is a pitch with a loading spinner. Before you build, write down what result would make you stop, and show that sentence to the room at the same time as the results. If nothing could have made you stop, you were not deciding, and the afternoon would have been better spent on the case set alone.

PART FIVE

What this does not solve

Three real limits, and one place where the prevailing view is right and I am not.

Twenty cases will not surface emergent multi-step failure. Trajectory behavior over long horizons, the loop that compounds, the retry that snowballs its own context, none of that appears in an afternoon. It appears in evaluation, in load, and sometimes only in production. The judgment prototype answers whether the decision is right. It says almost nothing about whether the system is reliable, and those are different questions that get confused constantly.

The expert can be wrong. They validate against their own judgment, which may be stale, idiosyncratic, or a habit that was never policy. A care-coordination agent once passed every check while reasoning from a clinical guideline that had been revised three years earlier, and everybody in the room agreed with it. Validation against an expert is a much better instrument than validation against nobody, and it is not a source of truth.

Cost and latency are invisible here. A prototype that makes forty model calls to reach a good answer looks identical to one that makes four, and only one of them has unit economics. That question belongs to the cost model, not to this artifact, and a good demo has hidden a bad business case more than once.

And the prevailing view is not simply wrong. A product manager who cannot build is slower, and speed compounds across a career. The correction is not that PMs should stop building. It is that at high autonomy the thing worth building is the case set rather than the interface, and the deliverable is the disagreement rather than the demo.

THE ONE LINE

The prototype is still the fastest way to decide. It just stopped being something you look at.

At the bottom of the ladder, build the screen, because the screen is the product. At the top, build twenty cases and put them in front of the person who used to make the call. What comes back is not feedback. It is the bounds, the escalation policy and the eval set, written by the only person who could have written them, in an afternoon, before anyone built the thing for real.

Sources and status. Drawn from the *Agentic AI for Product Leaders* series by Yoram Friedman: *Why Agentic AI Products Fail*, chapter 5 on prototyping to decide, chapter 4 on how the work splits and the two briefs, chapter 8 on the two kinds of human-in-the-loop, and chapter 9 on evals and the golden dataset; *Agentic AI for Busy Product Managers* (second edition) on the autonomy ladder, the four runtime artifacts and the departed user; and *The Agentic AI Team* on the seam between the person who decides a boundary and the person who enforces it. The observation that circulating prototype demos are overwhelmingly deterministic user interfaces is the author's, from the current discourse rather than from a survey. The care-coordination case is an illustrative composite assembled from documented guideline-drift patterns. Figures 1 and 2 are structural illustrations of the argument rather than measured data. The twenty-case count is a working default, not a threshold; the useful number is however many hard cases an expert will adjudicate in one sitting.