AI Makes Prototypes Cheap—and Product Judgment More Expensive

PM Daily Digest

2026-09-15

AI Makes Prototypes Cheap—and Product Judgment More Expensive

By PM Daily Digest • September 15, 2026

AI is making prototypes and software experiments dramatically cheaper, but the period's strongest signals point to a sharper divide between learning artifacts and reliable products. The practical response is stronger evidence standards, guarded context access, and clearer expectations for AI-era PM roles.

Big Ideas

Separate build-to-learn from build-to-earn. Teresa Torres and Petra distinguish cheap, disposable interactive prototypes from production software that customers pay for. The latter still requires maintainability, non-functional requirements, reuse, security, scalability, cost control, and skilled human oversight. [1] AI products add error analysis, meaningful evaluations, and prompt/orchestration iteration; Torres describes the first 60–70% as relatively easy prototype work, while the final 30% can take months to years. [1] **Apply:** label every artifact as learning or production before work starts, and do not let a successful prototype waive production gates.

Cheaper building raises the evidence bar. Hiten Shah argues that AI can make a weak idea look real—a polished site, working demo, and persuasive analysis—while changing little for customers; compliments, signups, repeated use, displaced behavior, payment, and retention are progressively stronger signals. [2] In Beam's research, more than 100 people replied and 42 documented their workflows; 29 interacted with another Mac several times a day, 28 kept it nearby, and only four usually wanted the whole computer while 18 wanted one application or a few specific ones. [2]

Tactical Playbook

Use AI as an internal context multiplier, not an unsupervised customer oracle. One workplace experiment connected Claude to server logs, code, Swagger, ticketing, and Sentry to answer impact and incident questions, then proposed a Slack/Teams or SaaS interface for non-technical colleagues. [3] A separate team found that exposing the codebase directly to support produced confident but incorrect customer answers, especially when answers required multiple repositories, business context, configuration/data, or judgment beyond the visible code. [4, 5] Start with three gates: connect the sources; use the agent to accelerate technical staff; then expand access only as documentation and permissions are good enough. [6, 7]

Validate replacement behavior, not enthusiasm. Match the research method to the unanswered question and its cost: conversations for premise checks, observation for workflow detail, manual fulfillment for willingness to pay, and software when real use will teach more than another discussion. [2] Once a product is in the workflow, ask, "If it were not installed, what would you have done instead?" Look for the old workaround to disappear, then measure repeat use, payment, and retention. [2] Your own pain is customer zero—not proof of a market; the idea still has to earn customer one by surviving contact with other people's workflows. [2]

Case Studies & Lessons

Superhuman is extending meeting notes into follow-on work. Superhuman says users had long requested a notetaker and that Fathom's category expertise helped drive the decision. Its stated opportunity is not merely capturing and summarizing meetings, but turning what was discussed, decided, and committed into useful work by combining meeting context with email, calendar, documents, and other workplace data. [8] The teams are building the roadmap in parallel, with a planned Superhuman Mail integration. [9] The product lesson: define the downstream job an AI artifact should trigger, then connect it to the systems where that job is completed.

Career Corner

Product-sense interviews reward deliberate narrowing. Aakash Gupta's rubric spans prioritization, user empathy, structured thinking, creativity, judgment under ambiguity, and metrics fluency. His recommended shape is to defend one segment, rank pains by severity $\times$ frequency, choose one solution while naming its trade-off, and define a success metric with a counter-metric; practice one question aloud, timed, each day. [10]

Scope the "AI PM" role before preparing for it. A current hiring discussion describes startups testing system design, API debugging, and model-latency understanding because small teams need contributors who can ship. [11] But

another practitioner argues that PMs should own the customer and business outcome, understand what technical evaluations measure and when they matter, and not be pushed into doing a data scientist's job. [12, 13] Ask what the role will own over the next year; if it is meant to discover the path through repeated pivots, hiring advice favors a creator-builder with product sense rather than a generic "AI PM" label. [14]

Tools & Resources

A lightweight agentic shipping review. The Product Compass guide points PMs to a free `pm-skills` repository and a practical sequence: compare product documentation with the codebase, audit unit/integration/end-to-end/live/CI tests, and request prioritized code, security, and performance findings. [15] Keep a human in the loop for common user flows: the guide notes that agents can miss dynamic or briefly visible UI elements, so manually clicking through those flows remains necessary. [15]

Sources

1. Delivery Isn't Free - All Things Product with Teresa & Petra
2. X article by @hnshah
3. r/ProductManagement post by u/Human-Possession135
4. r/ProductManagement comment by u/smashhuevo
5. r/ProductManagement comment by u/smashhuevo
6. r/ProductManagement comment by u/Pandas1104
7. r/ProductManagement comment by u/Resident-Athlete-268
8. X post by @shishirmehrotra
9. X post by @rahulvohra
10. substack
11. r/ProductManagement comment by u/GrowWithPokeBot
12. r/ProductManagement comment by u/HustlinInTheHall
13. r/ProductManagement comment by u/jrodicus100
14. r/ProductManagement comment by u/Mid0
15. Product Engineering for PMs, Part 3: Build a SaaS App Without Coding