Prototypes Replace PRDs as PMs Rework Discovery, Buy-In, and Team Ops

PM Daily Digest

2026-04-11

Prototypes Replace PRDs as PMs Rework Discovery, Buy-In, and Team Ops

By PM Daily Digest • April 11, 2026

This brief covers the shift from specs to working prototypes, a tighter discovery method built around measurable pain and forcing functions, and new tactics for stakeholder buy-in. It also includes case studies from OpenAI/Figma, Stripe, and the PM community, plus career and tooling takeaways.

Big Ideas

1) Prototypes are becoming the working language of product teams

Traditional product development was built around the expense of software, so teams climbed an artifact ladder—specs, wireframes, detailed designs, prototypes, then MVPs—to build conviction before investing [1]. Ravi Mehta argues AI changes that constraint: working software can now be produced fast enough that prototypes are becoming part of how teams communicate, decide, and validate throughout the lifecycle [1].

> "The prototype's job is to give the team a running start, not to cross the finish line." [1]

Why it matters - Handoffs can collapse into tighter loops as PM, design, and engineering work more like a jazz band than an assembly line [1]. - The economics of exploration have flipped: building several options is often cheaper than over-debating one [1]. - Prototype code is often disposable by design; in one audited AI-built prototype, only 30% of the code was salvageable for production [1].

How to apply - Pick the prototype type that matches the question: **concept** for direction, **design** for stakeholder alignment, **research** for usage validation, **technical** for feasibility [1]. - Prototype with a learning goal, not just speed; the

teams pulling ahead prototype constantly, but with a clear question attached [1]. - Normalize throwaway work. Anthropic reportedly cycles through 10 or more prototypes for a single feature, with each iteration compressed to hours [1].

2) Discovery needs a forcing function, not endless validation

A recurring community theme: teams get stuck because they chase consensus on what to build instead of clarity on which problem matters most [2]. The more practical framing is to filter for measurable pain, validate quickly, then make a bet [2].

> "Teams that stay in discovery forever are usually avoiding that moment, not lacking information." [2]

Why it matters - Minor or boring solutions often come from jumping to features before the underlying problem is sharp enough [2]. - Significant product can ship in 3-6 months, but only if there is an explicit moment when discovery ends and commitment starts [3, 2].

How to apply - Time-box phase 1 to 4-6 weeks: kill ideas that do not address measurable user or business cost, then stack-rank survivors by pain severity [2]. - Time-box phase 2 to 3-4 weeks: validate only the top 2-3 problems using interviews, usage data, and sales/support patterns; pick the option with the clearest signal, not perfect consensus [2]. - Keep validation cheap: interviews, prototypes, landing pages, and broad workflow conversations all surfaced as useful tactics [4, 5].

3) Buy-in is a product problem too

Across Tony Fadell and Strategyzer, the pattern is consistent: when data is incomplete or stakeholder alignment is weak, PMs need more than facts. They need narrative, visualization, and a plan for moving people toward participation [6, 7].

> "You tell a story." [6]

Why it matters - Fadell's point is not to ignore data; it is to show that you found the available data, understand the customer, have judgment, and can explain business impact when hard proof is absent [6]. - Strategyzer defines buy-in as a combination of evaluation and participation; without both, you do not really have buy-in [7]. - Misclassifying bystanders as opponents can create resistance that was not there to begin with [7].

How to apply - Use simple storyboards or value scenes to show the moment of need, today's workaround, and tomorrow's better state [7]. - Map stakeholders by evaluation and participation: bystanders, supporters, testers, objectors, blockers, champions, and so on [7]. - Focus first on allies and the persuadable

middle, then move people one step along the spectrum instead of trying to convert everyone at once [7].

4) Internal AI tools may become the next product pipeline

Andrew Chen's theory is that a large wave of AI-native products could emerge from internally built tools, especially those created by non-engineers and adopted across teams [8].

Why it matters - Internal teams can act as an immediate early-customer base; as Chen puts it, the organization itself can function like the network [8]. - If internal tools spread, get blogged, or are open-sourced, they can become startup seeds rather than one-off automations [8].

How to apply - Treat internal daily use as a signal worth instrumenting, not just an ops convenience [8]. - Watch for tools that move beyond one team and solve a repeatable workflow that other companies might share [8].

Tactical Playbook

1) Run an 8-week discovery reset

One of the clearest community answers came from a B2C PM with an established product and market share, but weak problem/solution consensus [3, 9].

1. **Start with every live idea, then filter hard.** In the first 4-6 weeks, cut anything that does not address measurable user or business cost [2].
2. **Rank pain, not feature excitement.** Stack-rank what survives by pain severity [2].
3. **Limit the field.** Take only the top 2-3 problems into validation [2].
4. **Use the fastest signals available.** Interviews, usage data, sales/support patterns, prototypes, and landing pages all appeared as acceptable fast-validation inputs [2, 4, 5].
5. **Pick the clearest signal.** Do not wait for perfect consensus [2].
6. **Create a forcing function.** Decide in advance when validation ends and building starts [2].

Why this works: It reduces the risk of both failure modes described in the thread—shipping nothing meaningful and shipping only minor enhancements that never excite anyone [3, 2].

2) Build buy-in when hard data is incomplete

1. **Write the *today* scene.** Show the moment of need, the current workaround, and why existing solutions are inadequate [7].
2. **Write the *tomorrow* scene.** Show how the proposed solution changes that situation in concrete terms [7].
3. **Make your judgment legible.** Show that you found the available data, understand the customer, and can explain business impact [6].

4. **Map stakeholders by evaluation and participation.** Separate bystanders, supporters, testers, objectors, blockers, champions, and saboteurs instead of treating everyone as either aligned or resistant [7].
5. **Start with allies and the persuadable middle.** Strategyzer's guidance is to move people one step, not all the way [7].
6. **Use short visual feedback loops.** A five-minute de Bono-style round—clarify, critique, like, improve—can surface more useful feedback than long unstructured debate [7].

Why this works: Fadell's point is that storytelling is what gets teams to take a leap of faith, while Strategyzer's point is that buy-in requires both positive evaluation and participation [6, 7].

3) Use the right prototyping stack for the stage

1. **Go wide in canvas** when exploring new directions or collaborating across the team [10].
2. **Switch to code** when you need to feel interactions, test responsiveness, or work with real data [10].
3. **Use both** for last-mile polish and shipping; one source says a round-trip that used to take a sprint can take ten minutes [10].
4. **Adopt in a low-risk order.** Start with polish, not a full process rewrite [10].
5. **Prove the loop once.** Import one screen from code to Figma, change it, push it back, and verify it works before scaling [10].
6. **Move earlier once comfortable.** The reported payoff is earlier edge-case detection and strategy discussions that start with working software instead of static decks [10].
7. **Use AI as a tutor.** Ask the system to explain architecture, page structure, and redundancy as you learn [10].

Why this works: It lets PMs add real prototyping capacity without replacing the entire workflow on day one [10].

4) Package each release from one source of truth

1. **Write the core update once** instead of rewriting the same sprint summary for five audiences [11, 12].
2. **Create templates** for release notes, exec updates, CS briefs, emails, and Confluence pages [12, 13].
3. **Store instructions with the agent.** Include tone, purpose, required formats, and source locations [13].
4. **Point the agent to your source system**—an MD file, Notion, Confluence, or repo [13].
5. **Iterate a v1 quickly and improve over time.** The suggested setup cost was only a few hours for a first version [13].

Why this works: One PM estimated release packaging alone consumed about

half a day in a two-week sprint; templated AI support turns that into repeatable overhead instead of recurring drag [11, 13].

Case Studies & Lessons

1) OpenAI and Figma: running software becomes the alignment artifact

> "The phrase inside OpenAI - prototypes, not PRDs." [10]

Inside this workflow, PMs bring working prototypes to design reviews and ship PRs to stress-test ideas; content designers are also submitting PRs, and the Codex/Figma loop enables high-fidelity movement between code and canvas [10, 14]. The broader model is not role collapse but tool convergence: designers can ship code, PMs can prototype, engineers can contribute to design systems, while each role keeps its own core question [14, 10].

Lesson: The reported benefit is earlier edge-case discovery and immediate feedback because the thing exists [10].

2) Stripe machine payments: small-N traction, tight preview

Stripe said machine payments are already seeing consistent, real daily use across a number of businesses, albeit with a very small N [15]. Some implementations are powered by Tempo and are working in production [15]. Stripe has kept the product in private preview to go deep with partners on use cases while refining its APIs, alongside published machine payments docs [15].

Lesson: Early daily usage plus a constrained partner program can be a better signal than broad availability when the workflow and API surface are still forming [15].

3) Validation before code: roadmap-first feedback

One founder described spending months shipping features that got almost no usage or feedback [16]. The process changed after reversing the order: break the product into features and ideas, share it as a simple roadmap, and let users react, request, and vote before building [16]. The result, in the founder's words, was feedback before effort rather than after [16].

Lesson: When signal is weak, a lightweight roadmap can function as a validation artifact before engineering work starts [16].

4) DoorDash's Team OS: AI leverage comes from structured context

Aakash Gupta highlighted Hannah Stulberg's Team OS at DoorDash: a shared system of specs, code standards, playbooks, and other structured context that Claude can navigate efficiently [17]. In the example, a customer query used only 3% of the context window, a non-technical strategy partner was submitting PRs

every day, and the claimed math was 2 hours of setup per person for 5+ hours saved per week per person—50+ hours weekly on a 10-person team [17].

Lesson: AI output quality is not just a model question; it depends on whether the team has turned its operating context into something machines can reliably retrieve [17].

Career Corner

1) Build adjacent fluency, but keep your PM spike

Tool convergence does not erase roles. In the OpenAI/Figma framing, engineers ask how to build well, designers ask how the experience should feel, and PMs ask why it should be built at all [10]. At the same time, PMs can use design skills to prototype flows, and AI can act as a patient tutor as they learn code and architecture concepts [10].

How to apply - Build something small for yourself to get reps; examples cited included non-engineers building an iOS app or a drag-and-drop HTML tool after simply downloading the app [10, 14]. - Treat the tool skill as an amplifier for judgment, not a replacement for problem selection and prioritization [10].

2) When evaluating PM orgs, ask how signal actually flows

A Reddit discussion surfaced a useful tension. One Head of Product emphasized organization as the key PM trait and said PMs often get customer feedback through commercial teams [18]. Some responses said that can be normal in regulated environments like healthcare, where direct access is limited [19]. Others saw it as a possible sign that product does not really lead strategy [20]. A related counterpoint was that customer obsession, agency, and taste still matter; organization is essential, but not sufficient [18, 21].

How to apply - In interviews, ask how qualitative and quantitative signal reaches PMs, whether there is a feedback loop, and how much roadmap authority the product team actually holds [19, 21, 20]. - Do not treat direct interviews as the only valid input channel, but do treat weak feedback loops as a real risk [21, 20].

3) If the job is all packaging and busy work, treat that as career data

One PM described a role with no roadmap ownership, no dev interaction, and no shipped features after a year, leaving mainly requirement writing and ignored recommendations [22]. Another thread described a different execution tax: roughly half a day per sprint spent repackaging the same release information for different audiences [11]. The consistent advice was pragmatic: focus on what you can control, land one recruiter-ready accomplishment if possible, and explore better roles rather than waiting indefinitely [23, 24, 25].

How to apply - Audit whether the role is increasing your leverage or just your admin load [22, 11]. - If you leave after about a year, explain the constraint honestly and point to the clearest thing you improved or shipped [26, 24].

Tools & Resources

1) Codex desktop app + Figma MCP

Why explore it: This is the clearest example in the notes of a high-fidelity code canvas loop. It supports importing running code into Figma, editing there, and pushing changes back to code [10].

Best first use: Do one end-to-end loop on an existing project before you try to redesign your process [10].

2) Prototype decision matrix

Why explore it: The four-type taxonomy—concept, design, research, technical—gives PMs a simple way to choose the right artifact for the right uncertainty [1].

Best first use: Add the prototype type to your next discovery plan so the team knows what question each artifact is supposed to answer [1].

3) Strategyzer's workshop set

Why explore it: The combination of customer ecosystem mapping, value scenes, and the 9 Personas of Change turns abstract discussion into concrete artifacts [7].

Best first use: Run a 20-30 minute ecosystem map for a complex B2B problem, then use a five-minute feedback round to sharpen the proposed change story [7].

4) Team OS

Why explore it: Shared AI-readable context can reduce repeated explanation, improve retrieval quality, and let non-technical teammates contribute more directly [17].

Best first use: Start with one repo or workspace containing specs, standards, and indexed team playbooks instead of trying to structure everything at once [17].

5) AI release-comms kit

Why explore it: A lightweight stack of instruction files, templates, examples, and one source of release truth can cut recurring packaging work [12, 13].

Best first use: Build templates for release notes, exec updates, CS briefs, and Confluence pages, then iterate them every sprint [13].

Sources

1. The product lifecycle is broken
2. r/ProductManagement comment by u/ProductManagement3
3. r/ProductManagement post by u/Ill_Show6713
4. r/ProductManagement comment by u/Enough_Big4191
5. r/ProductManagement comment by u/intentions_are_high
6. X post by @tfadell
7. Drive impact across your business - Strategyzer's next chapter
8. X post by @andrewchen
9. r/ProductManagement comment by u/Ill_Show6713
10. How to Design like OpenAI and Figma
11. r/ProductManagement post by u/MundanePassage2201
12. r/ProductManagement comment by u/beth_maloney
13. r/ProductManagement comment by u/beth_maloney
14. substack
15. X post by @patrickc
16. r/startups post by u/d_uk3
17. substack
18. r/ProductManagement post by u/Uncomfortabl
19. r/ProductManagement comment by u/No-Solid-4255
20. r/ProductManagement comment by u/Uncomfortabl
21. r/ProductManagement comment by u/bien-fait
22. r/ProductManagement post by u/noobies123
23. r/ProductManagement comment by u/Broad-Permit-7355
24. r/ProductManagement comment by u/LatterLuck8678
25. r/ProductManagement comment by u/Broad-Permit-7355
26. r/ProductManagement comment by u/noobies123