We're all becoming technical product managers

We killed the concept of squad/pod over a year ago. We let go of junior engineers and product designers. All remaining engineers became product engineers, responsible for sections of our product and owning KPIs. That was a good move. But there is more to come.

Since then, every non-engineer on our team has vibe-coded or automated multiple processes across marketing, research, and customer service. This has allowed us to move faster. But so it's been the case for our competitors. And this is just the beginning.

As a technical founder at a product-led growth company, I've served as the de facto product manager. As a product manager, I spent most of my time applying the scientific method to our product: observing users, ideating, prioritizing, hypothesizing, communicating with engineers, coordinating work, and analyzing. Each idea became an experiment. Each experiment got an engineer. We would work on dozens of experiments in parallel.

The main bottlenecks were development costs and development speed. Thomas Edison said that "innovation is 1% ideation and 99% perspiration", but AI is changing that on its head. The cost and speed of software development are converging to 0. The bottleneck is being removed. As engineers build features faster and non-engineers build features on their own, I will soon become the bottleneck.

That is, unless we all become product managers. Technical product managers, to be exact. And that's where we are heading:

Most members of our team, not just engineers, will soon be **technical product managers (TPM)**. I'll oversee our product, and each TPM will oversee a portion of it.

You'll be responsible for:

- Understanding users, ideating, prioritizing, hypothesizing, communicating, influencing architecture, and analyzing the outcomes.
- Ensuring your product works, SLAs are met, and goals are achieved.
- For coordinating with me, with AI, and with other technical product managers.

Engineers will get the training and toolset required to better understand the "business".

Non-engineers will learn software engineering. Yes. What some members of our team invested 4 to 10 years of their lives learning in college, you'll have to learn quickly. It will be challenging. Some people compare building startups to building a plane while flying it. Thanks to this transition, the analogy now includes learning to build the plane simultaneously, too.

Everyone will get the frameworks and harnesses required to build scalable, reliable software, as well as the budget and tools needed to have agents develop, test, release, test again, identify some bugs, and fix them 24x7.

The transition to TPMs won't be easy for all. It will require hard and smart work. Opening our minds. Making sacrifices. Some will love it. Some may not enjoy it. Some may not be capable at all. The ones who succeed will ride the AI wave (either here or somewhere else). The ones who don't will miss this once-in-a-lifetime opportunity.

But so that we are aligned, this is what we'll expect from you:

- A good TPM is a `generalist`.
- A good TPM is `obsessed with outcomes`. They have clarity about their KPIs. They know others' KPIs and the business's north-star metrics. A bad TPM pushes in silos, unaware of its impact on other areas.
- A good TPM `understands the user`. Empathizes with their pains and needs. A bad TPM hides behind technological limitations to justify offering a subpar experience to our users.
- A good TPM `generates many ideas`, both good and bad, and doesn't take it personally when their ideas are criticized. They know that it is not about being right but about getting it right. A bad TPM doesn't ideate frequently or doesn't share their ideas with others.
- A good TPM `has an intuitive sense of prioritization`. They've considered so many ideas and their ICE (impact, cost, and effort required) that it is easy for them to determine what should be next. A bad TPM relies on others to determine what's next. `They mistake speed for direction.`
- A good TPM `knows how to measure success`. They are mindful of which product and engineering metrics shall be impacted by each experiment. They are obsessed with split a/b tests and feature flagging. A bad TPM pushed functionalities live, without clarity on what it means for a hypothesis to be proven valid.
- A good TPM `abstracts and expects abstraction`. They don't allow functionality to be duplicated unintentionally. A bad TPM creates more technical debt than the debt they pay.
- A good TPM `defines comprehensive test scenarios` (a.k.a. "evals"). They make sure main and edge cases are properly tested. A bad TPM is constantly chasing bugs.
- A good TPM `engineers software without typing code`. They can read code and understand code that AI wrote. They get AI to do what's needed, not by coding it themselves, but by tweaking the architecture and test scenarios so that the AI figures out what to do. A good TPM doesn't give fish to AI. It teaches AI to fish. `A bad TPM vibe codes`.
- A good TPM **shows** `OCD-level attention to detail`. Whether it is visual, architectural, or operational, good TPMs care about the details that affect users' experience. Bad TPMs rely on others to identify issues.
- A good TPM has **a** `spider sense`. They intuitively know when things don't add up. A sense that makes them go deep into an issue and find out inconsistencies/discrepancies. A bad TPM doesn't detect bullshit and is tricked by the overconfidence of AI.
- A good TPM **is a** `team player`. They are aware of the roadmap of all other TPMs. Proactively finds opportunities to collaborate. Raise their voice when they need help. A bad TPM works in isolation.
- A good TPM is a `solid communicator`. They provide enough context for both humans and AI agents. They always err on the side of overcommunicating.
- A good TPM `knows how to manage AI agents`. They know what tools to harness them with and what rail guards to give them.
- A good TPM is `mindful of token costing`. They estimate the cost of creating, maintaining, and hosting new functionalities. A bad TPM only realizes how much something costs when it's brought to their attention for being too expensive.
- A good TPM is `always in Kaizen mode`. Looking for areas of improvement is natural to them. They are

proactive. A bad TPM is reactive.

- A good TPM `follows our standards and influences them`**.** They understand that the requirements and bureaucracy we have in place are needed for the cohesiveness of our well-oiled machine. A bad TPM breaks our rules unintentionally.
- A good TPM `earns my trust`. A good TPM is good enough so that I don't have to sign off on all of their work, but also knows when it's appropriate to consult with me. A bad TPM requires my handholding or requires my proactive attention to correct the wrong path they've taken.
- A good TPM `learns fast`**.** They are always learning and sharing what they learn with others. Bad TPMs rely on others to pull them and don't share what they learned.
- A good TPM `ships fast` **and is always iterating.**
- But most importantly, good TPMs **create value faster than our competitors.** To quote Daft Punk, they are and make us "*Better*, *Faster*, *Stronger*."

What's next for each one of us? Identify the areas where you are strong (ask others for their opinions) and share your knowledge. Identify the areas where you're weak and ask others for materials and coaching. And ship, ship, ship.