LEANSpark

THE AI-NATIVE PLAYBOOKS · Nº 7

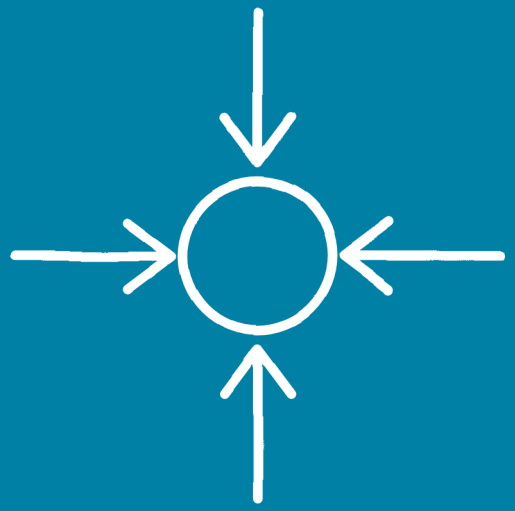
How to Design What Customers Want

The 5P offer: turning what customers told you into the thing they'll actually switch for

LEANSTACK · Ash Maurya

leanspark.ai

Contents



CHAPTER 1

An offer, not a product

A product is what you build. An offer is what someone buys. Design the second one first.

CHAPTER 1

An offer, not a product

You've done the interviews. You know the problem is real, you can hear your segment's voice in your sleep, and every instinct says it's time to design the product. Here's the most common failure mode at exactly this moment: founders design a *product* when they should be designing an *offer*.

A product is what you build. An offer is what someone buys. They're not the same thing, and the difference decides whether anything you build next gets bought.

Consider a CRM that automates follow-up with cold leads. The product pitch: *"AI-powered CRM with sentiment-aware follow-up sequences, integrations with 47 tools, and advanced pipeline analytics."* The offer pitch: *"Stop losing deals to late or forgotten follow-up. Double your close rate on stale pipeline in 90 days, or don't pay us."* Same underlying system. One is a feature list. The other is a committed transformation.

> **Why features don't move purchases.** A purchase happens when **Push + Pull > Inertia + Friction**. Features contribute to Pull, but only weakly, because features are *attributes* of the new way, not *outcomes* for the buyer. Prospects don't reach for a better feature list. They reach for a better situation. The offer has to describe the situation, not the system.

The offer you're about to design has five parts, the **5P Matrix: Problem**, **Promise**, **Proof**, **Product**, **Price**. Notice that Product is the *fourth* P, not the first. That's deliberate. Most founders start with Product and try to retrofit the rest around it. It rarely lands. Start with the segment's real problem and design backwards from there.

The good news: the hard work is already done. Your interviews hold the transformation in the segment's own words. Designing the offer isn't inventing from scratch. It's translating specific evidence into a five-part specification the prospect will recognize when you show it to them.

A useful check as you draft each P: read it aloud and ask, *"Would a prospect from my segment nod at this, or politely say 'that's interesting'?"* A nod means the offer speaks their language. "That's interesting" means you're still in feature-land, and the prospect is being polite. Revise.

Hear your idea both ways

The job: hear the difference between your product pitch and your offer pitch.

IN LEANSPARK — YOU TYPE

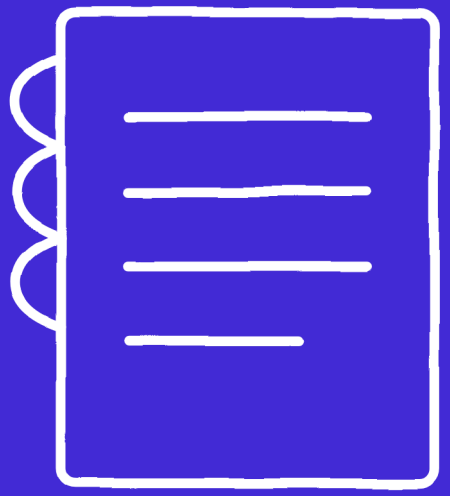
Rewrite my pitch as an offer

Writes your idea both ways — the product pitch (features and capabilities) and the offer pitch (a specific, committed transformation in your segment's words) — then runs the nod test on the offer version against your interview evidence, line by line.

RETURNS Both pitches side by side and a nod-test read: which lines a real prospect would nod at, and which are still feature-land. The offer version seeds the five Ps you're about to draft.

VIA MCP — PROMPT ANY AGENT

"Read `business_model://current` . Write my idea two ways — a product pitch (features) and an offer pitch (a specific, committed transformation for my segment) — then judge the offer version against my interview evidence: would a prospect nod, or politely say "that's interesting"?"



CHAPTER 2

Mix the feature cocktail

Delighters first, then performance, then must-haves. If a feature doesn't trace back to your interviews, it's a guess.

CHAPTER 2

Mix the feature cocktail

Before drafting the five Ps, you need to know which features actually cause a switch. In the 1980s, quality researcher Noriaki Kano plotted customer satisfaction against investment in a feature and found something most roadmaps still ignore: features don't all behave the same way, and investing more in the wrong kind buys you nothing.

FEATURE TYPE	HOW SATISFACTION RESPONDS	CAR EXAMPLE
Must-have	Absence makes customers unhappy; presence is merely expected.	Brakes
Performance	Linear: more is better, less is worse. Customers compare on these.	Horsepower
Delighter	Unexpected. Presence creates off-scale happiness; absence costs nothing.	Park-assist
Indifferent	No effect either way.	The cabin air filter's color
Reverse	Negative: more is worse.	Emissions

> **Delighters decay.** A feature's ability to delight has a finite lifetime. Multi-touch was a delighter when the iPhone revealed it; today it's a must-have on any phone. This cuts both ways: copying the incumbent's old wow-features buys you nothing, and your own delighters won't stay delighters forever. One more reason speed matters.

A startup is constrained on speed, spend, and scope, so the cocktail runs opposite to most founders' instincts: **delighters first, then performance, then must-haves.** Must-haves alone lose to the incumbent, because their presence is expected and the incumbent has them plus all the inertia of staying put. Performance features cause a switch only at 3x to 10x better, an expensive head-on race against a better-resourced player. Delighters are the startup's edge: they don't exist in the incumbent's offering, so they're unique by definition, and unique plus off-scale happiness is a textbook unique value proposition.

How do you pick the right ones? Many founders guess. You don't have to, because your interviews already produced the evidence: the **job requirements** your prospects revealed every time they hired or fired a solution.

- **Firing criteria** (why alternatives lost) mark the must-haves your solution cannot fumble.
- **Hiring criteria** (why the winner won) mark the performance dimensions your segment actively compares. Compete here only where you can be dramatically better.
- **Tradeoffs** (shortfalls prospects accepted anyway) mark the whitespace. A solution that removes a tradeoff everyone else forces is unexpected by definition: a delighter candidate, grounded in evidence instead of imagination.

If a feature on your list doesn't trace back to a hiring criterion, a firing criterion, or a tradeoff, it's a guess. Cut it or park it.

> **New to this? Each card gives you two ways to run it.** Type the **In LEANSpark** line into the chat, or use the **Via MCP** prompt inside your own AI assistant. LEANSpark does the assembly and the scoring; the design judgment stays yours.

Assemble your job requirements and mix the cocktail

The job: turn 20+ interviews into the evidence-backed feature mix that causes a switch.

IN LEANSPARK — YOU TYPE

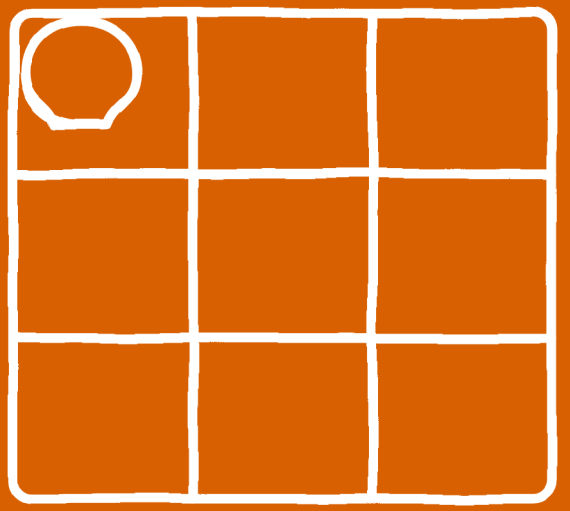
Assemble my job requirements

Sweeps your Customer Forces Stories, extracts hiring criteria, firing criteria, and tradeoffs for every alternative in each consideration set, ranks them by frequency, then maps them onto the Kano model — must-haves you can't fumble, performance dimensions worth 3x, and the tolerated tradeoff that's your best delighter candidate.

RETURNS Ranked job-requirement tables and a proposed feature cocktail, each line traced to the interviews that back it. You choose the delighter to lead with.

VIA MCP — PROMPT ANY AGENT

"Read `business_model://current` and my recent interviews. Assemble job requirements — hiring criteria, firing criteria, tradeoffs — per consideration set, ranked by frequency. Map them onto the Kano model and propose my feature cocktail: one delighter, the performance dimensions that matter, the non-negotiable must-haves. Flag any feature I've mentioned that traces to none of them."



CHAPTER 3

Problem and Promise

State their situation better than they can. Then name the after-state so specifically it becomes believable.

CHAPTER 3

Problem and Promise

The first two Ps are a before-and-after pair. **Problem** is where the prospect is today. **Promise** is where they end up. Get the pair right and the other three Ps get easier; get it wrong and nothing else in the offer lands.

Problem: say it better than they can

You're articulating Problem well when a prospect reads it and says some version of *"you described my situation better than I could."* That's the bar. Prospects don't recognize "inefficient reporting" as their problem. They recognize "I spend 4 hours every Monday morning stitching data together before the weekly team meeting, and then I'm ten minutes late to it again."

The raw material is in your interviews. Re-read your Customer Forces Stories back to back, highlighter in hand, and mark every phrase where a prospect described their situation in concrete, emotionally loaded language: specific moments, emotional words (*drowning*, *dreading*, *scrambling*), measurable cost, and phrases multiple prospects repeated. The Problem articulation is a synthesis of those phrases into a paragraph that sounds like one specific person from your segment talking about their life. Keep it to 80–120 words.

> **The three failure modes.** *Too abstract* ("CS leaders struggle to scale training") could describe anyone. *Too generic* ("training new hires is hard") belongs to no segment. *Too mechanical* ("43% lower adoption than needed") is your metric, not theirs. The fix is the same for all three: go back to the transcripts and copy phrases in.

Promise: the transformation, not the features

The Promise is the specific after-state your offer delivers. Four properties make it strong: **specific** (an outcome, not an adjective), **measurable** ("cut ramp time from 6 weeks to 3" beats "faster ramp"), **time-bounded** ("within 90 days" tells them when it becomes true), and **emotionally resonant** (it connects to the outcome your interviews surfaced: confidence, control, freedom from dread). Miss any of the four and the Promise reads like marketing copy instead of a commitment. Keep it to 30–60 words; the prospect has to *remember* it after the pitch.

You'll be tempted to follow the Promise with "...because we provide AI-driven training paths, auto-assessment..." Don't. The Promise stands on its own. Features belong in Product, and mostly shouldn't surface until the prospect has committed to exploring the Promise.

Specificity is friction reduction. “Transform your team’s onboarding” creates weak Pull and high Friction: there’s nothing to believe or disbelieve, so it sounds like vapor. “Cut new-hire ramp from 6 weeks to 3 in 90 days, or you don’t pay” creates a concrete, falsifiable test — which, paradoxically, *lowers* the prospect’s Friction, because it moves the risk onto you.

Draft your Problem and Promise from real quotes

The job: write the before-and-after pair in your segment’s own words, not yours.

› IN LEANSPARK — YOU TYPE

Draft my Problem and Promise

Mines your Customer Forces Stories for the concrete, emotionally loaded phrases prospects actually used, synthesizes an 80–120 word Problem articulation in their voice, then ladders up to a 30–60 word Promise that is specific, measurable, time-bounded, and tied to the Pull your interviews surfaced — and runs the nod test on both.

RETURNS A drafted Problem and Promise, each phrase traced to the interview it came from, with the weak spots flagged. You judge whether a real prospect would nod.

› VIA MCP — PROMPT ANY AGENT

"Read `business_model://current` and my recent interviews. Draft the first two Ps of my offer: an 80–120 word Problem articulation built from direct prospect quotes, and a 30–60 word Promise that is specific, measurable, time-bounded, and emotionally resonant. Quote the interview lines behind each claim — don’t invent pain I didn’t hear."

CHAPTER 4

Proof

Prospects don't doubt the Promise in general. They doubt it for their situation, from you. Answer all three doubts.

CHAPTER 4

Proof

The third P is **Proof**: the evidence that convinces a prospect the Promise will come true for *them*. Proof is the single largest Friction-reducer in the offer. Get it right and prospects move fast. Get it wrong and they "think about it" forever.

Prospects don't reject offers because they doubt the Promise in general. They doubt *this specific Promise, for their specific situation, as delivered by this specific vendor.* Proof has to answer all three: the transformation is possible, it works for people like them, and your solution is the mechanism that delivers it.

Four kinds of Proof, in order of power:

1. **Paying customers.** By far the strongest, and at this stage you almost certainly don't have them. That's fine; generating the first ones is the point of the next playbook.
2. **Signed commitments or pilots.** What a well-run offer campaign ends with. Early pilots become Proof for the prospects after them.
3. **Data from adjacent work.** "I've delivered this transformation before, in a similar context, and here's what the numbers looked like." Specific numbers carry weight. Vague credentials don't.
4. **The mechanism.** "Here's why this works, and here's the path from where you are to where the Promise delivers." The weakest form, but often all you have early, and done well it's still meaningful: prospects who see a plausible mechanism accept the rest on credibility. Prospects who don't will not be moved by any number of logos.

> **What doesn't proof.** A 1,000-person waitlist (waitlist ≠ paying; it usually drags credibility down). Logos of customers you don't actually have (any hint of inflation collapses trust instantly). Generic industry statistics ("studies show 90% of teams struggle…" — it's not about *them*). Endorsements from non-customers ("my advisor loves it").

Proof has an implicit second layer: why *you*, now. One or two sentences on the specific experience or insight that makes this Promise achievable from your hands. Not credentials as flex; credentials as evidence you can deliver. If you struggle to articulate why you're the right one to deliver the Promise, you may not be. That's a signal worth sitting with.

Build your proof inventory

The job: find the strongest Proof you honestly have, and the doubts it leaves open.

IN LEANSPARK — YOU TYPE

Build my proof inventory

Inventories your available Proof by the four kinds — paying customers, signed pilots, adjacent-work data, mechanism — maps each item against the three doubts (possible at all, for people like them, through you), and flags anything on the anti-proof list before a prospect does.

RETURNS A ranked proof inventory, the doubt each item answers, the gaps still open, and a drafted one-or-two sentence “why you, now” stance.

VIA MCP — PROMPT ANY AGENT

"Read `business_model://current` and `offer://current`. Inventory my Proof by the four kinds in order of power, map each item to the three doubts it answers, flag anything that reads as anti-proof (waitlists, borrowed logos, generic statistics, non-customer endorsements), and draft my “why you, now” stance."

CHAPTER 5

Product and Price

Product is the minimum mechanism the Promise requires. Price is a fraction of the value, not a multiple of your costs.

CHAPTER 5

Product and Price

Product: features on a leash

The fourth P is **Product**, and here features finally enter the conversation. But they enter on a leash: Product is defined by *what the Promise requires*, not the other way around. For each element of the Promise, ask what's the minimum thing that has to exist for it to come true. That's the product definition. Everything beyond it either doesn't belong in this offer or belongs in a later cycle.

The right instinct is to strip Product *down*. Every feature you describe is a feature the prospect expects you to deliver; each one adds to their evaluation and your build cost. Your feature cocktail from Chapter 2 keeps this honest: the Product is the delighter, the one or two performance dimensions your hiring criteria say matter, and the must-haves your firing criteria say are non-negotiable. Everything else waits.

> **Deliver it as a service first.** Product also answers how the value is delivered: self-serve software, managed service, pure service, or hybrid. Early on, a service almost always beats pure software. You can deliver a service before the software is built, making the Promise real while the product develops — and managed delivery puts you inside the prospect's workflow, so you see what the software actually needs to automate instead of guessing.

The stripe-it-down test: *"If I could only build one thing in the next 30 days, which single element delivers the biggest chunk of the Promise?"* That one thing is the MVP of the offer. In the end, Product is three to five sentences about what's delivered, how, and why it maps to the Promise. It is not a feature list, an architecture, or a 12-month roadmap.

Price: designed, not discovered

The fifth P is **Price**, the element founders most commonly over-agonize, under-research, and set on a gut feeling that's nearly always wrong. The default instinct is to price from cost: development hours, hosting, a modest margin. Backwards. The prospect doesn't care what it cost you to build. They care what the Promise is worth to them. Start from value and work back.

From your interviews you can answer: what does *not* solving this cost the prospect, in time, money, or credibility? Convert the strongest-expressed cost to a number. If slow ramp costs a team roughly $40k per cohort, that's the anchor. Pricing at 10–20% of the value delivered leaves the prospect a clear win; above 30% the decision gets hard; above 50% prospects stall. For the $40k example, that's a defensible $4k–$8k.

> **One anchor, clearly.** Anchor against the cost of the status quo (“you’re already losing $40k per cohort; this is $6k”), or a pricier existing alternative, or a round credible number when the value argument carries itself. Pick one. Multiple anchors read as equivocation.

The “feels expensive” test: read your Price aloud. If it feels uncomfortable, if your instinct is to apologize for it, it’s probably about right. Prices that feel comfortable to the founder are usually too low, and underpricing telegraphs that you don’t believe your own Promise.

Define the minimum Product and design the Price

The job: scope the smallest mechanism that delivers the Promise, and price it to the value.

IN LEANSPARK — YOU TYPE

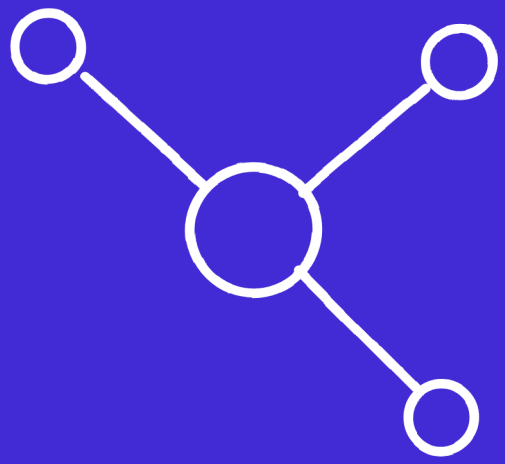
Design my Product and Price

Derives the minimum Product from your Promise and feature cocktail — delivery mode included, with a service-first option — then computes the value anchor from the costs your interviews surfaced and proposes a Price in the 10–20% band, in three structures: flat, per-unit, and outcome-based with the risk reversed.

RETURNS A three-to-five sentence Product definition, a stripped MVP-of-the-offer call, and a priced exchange with its anchor stated — plus the “feels expensive” read.

VIA MCP — PROMPT ANY AGENT

"Read `business_model://current`, `offer://current`, and `competitive://current`. Define the minimum Product that delivers my Promise (consider a service-first delivery), then price it: quantify what not solving the problem costs my segment from the interview evidence, anchor to it, and propose flat, per-unit, and outcome-based structures in the 10–20%-of-value band."



CHAPTER 6

One page, one story

Five good fragments don't add up to an offer. Integrate, read it aloud, and only then go pitch it.

CHAPTER 6

One page, one story

You have drafts of all five Ps. The last step is pulling them onto a single page and running the checks that separate a strong offer from five good-looking fragments that don't add up. The full offer, on paper, before any code or demo gets built: Problem (80–120 words), Promise (30–60), Proof (50–100), Product (50–100), Price (20–40). If it doesn't fit on one page, the offer isn't tight enough.

Four integration checks, in order:

1. **Does Promise solve Problem?** Read them side by side. If the Problem is about feeling like the bottleneck and the Promise is about reporting improvements, they don't hook together, and the prospect won't feel the offer was written for them.
2. **Does Product deliver Promise?** Walk the causal chain honestly: *"If I sold this tomorrow, could I deliver the Promise in the timeframe it names?"* If not, shrink the Promise or expand the Product. Don't pretend the chain works when it doesn't.
3. **Is Proof credible at your stage?** If you can't say your Proof out loud without a pang of "that's a stretch," tighten the Promise until existing Proof covers it, or find stronger Proof.
4. **Does Price anchor to the value?** $500 against a $40k problem makes prospects wonder what's wrong with it; $30k against $40k leaves less than a 2x win and stalls the decision. The Price should feel like obvious-win arithmetic, not a negotiation.

> **The aloud-read.** Read the whole page in order: *"You [Problem]. We [Promise]. Here's why [Proof]. We deliver this by [Product]. The exchange is [Price]."* That's the spine of a pitch. If it tells one coherent story aloud, you're ready. If there's a gap anywhere, you'll feel it in your own voice.

Run the integration check on your one-page offer

The job: catch the gaps between your five Ps before a prospect does.

IN LEANSPARK — YOU TYPE

Check my 5P offer

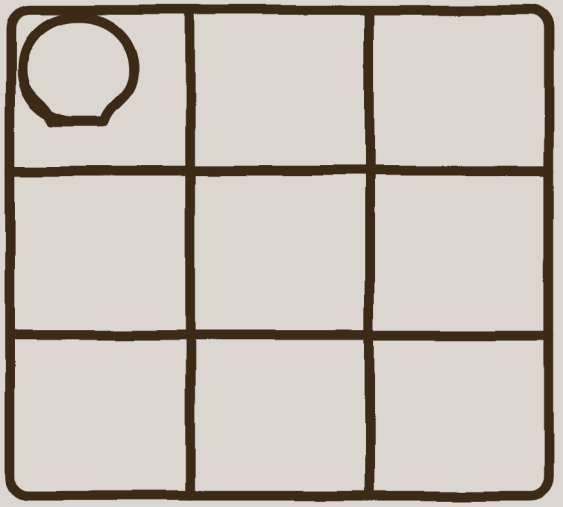
Assembles your five Ps onto one page, runs the four integration checks — Promise-solves-Problem, Product-delivers-Promise, Proof-credible-at-stage, Price-anchored-to-value — scores each, and renders the aloud-read spine so you can hear whether it's one story or five fragments.

RETURNS A one-page 5P Matrix with each check passed or flagged, the specific gap named, and the smallest revision that closes it. When all four pass, you're ready to build the demo on top of it.

VIA MCP — PROMPT ANY AGENT

"Read `offer://current` and `business_model://current`. Assemble my 5P Matrix on one page and run the four integration checks. For any check that fails, name the gap and propose the smallest revision — shrink the Promise before expanding the Product. Then write the aloud-read spine."

Everything from here is delivery. The one-page offer you just integrated is exactly what **How to Sell Before You Build** (No. 5 in this series) turns into a four-act pitch, a smallest-viable demo, and a set of real commitments. The demo isn't a product tour; it's a tour of these five Ps, made visual. Design here, sell there.



CHAPTER 7

Resources

Run your offer design on LEANSpark. Go deeper on the method.

CHAPTER 7

Resources

Design your offer on LEANSpark

— **Assemble the evidence** — job requirements extracted from your real interviews: hiring criteria, firing criteria, and the tolerated tradeoffs that hide your delighter.

— **Mix the cocktail** — an evidence-backed feature mix, with every guess flagged as a guess.

— **Draft the five Ps** — Problem and Promise written from prospect quotes, Proof matched to your stage, Product stripped to the minimum mechanism, Price anchored to the value.

— **Check the integration** — the four checks and the aloud-read, so the page tells one story before you pitch it.

— **Hand off to the pitch** — the finished 5P offer feeds straight into the Mafia Offer pitch generator from *How to Sell Before You Build*.

Design your offer at leanspark.ai.

Go deeper on the method

— **How to Talk to Customers** (No. 3 in this series) — the interviews this playbook spends. Do that one first.

— **How to Sell Before You Build** (No. 5 in this series) — pitching the offer you just designed, and counting who commits.

— *Running Lean* and **The AI-Native Founder's Playbook** (No. 1) — where Solution Design sits in the journey from idea to traction.

— The Demand Validation Playbook — the full course track behind this playbook.

The offer at a glance

P	WHAT IT IS
Problem	Their situation, in their words, better than they can say it. 80–120 words.
Promise	The after-state: specific, measurable, time-bounded, resonant. 30–60 words.
Proof	Why it's true for *them*, from *you* — mechanism first, pilots as they land.
Product	The minimum mechanism the Promise requires. Features on a leash.
Price	A fraction of the value, one clear anchor, uncomfortable to say aloud.

LEANSpark

How to Design What Customers Want is part of The AI-Native Playbooks, a series from LEANSTACK / Ash Maurya. LEANSpark turns your interviews into an offer worth switching for.

leanspark.ai