A STRATEGIC GUIDE

The Product Manager's Playbook

Building, scaling and leading products in the age of AI

SECOND EDITION



Robert Merrell
Founder, Product Advisory

P+ Product Advisory

Contents

Preface to the Second Edition

How to Use This Book

PART 1: FOUNDATIONS

Chapter 1. The Product Leader's Job, Now

Chapter 2. Finding Problems Worth Solving

PART 2: STRATEGY

Chapter 4. Vision, Strategy and Positioning

Chapter 5. Prioritization and Outcome Roadmaps

Chapter 6. The Numbers: P&L, Unit Economics and Pricing

Chapter 7. Managing a Product Portfolio

PART 3: BUILDING

Chapter 8. Building and Leading the Product Team

PART 4: CUSTOMERS

Chapter 12. Designing with Empathy

Chapter 13. Go-to-Market and Launch

Chapter 14. Customer Success and Retention

PART 5: GROWTH

Chapter 15. Partnerships and Ecosystems

Chapter 16. Scaling the Product and the Company

Chapter 18. The Integrated Playbook

Appendix: Templates

Preface to the Second Edition

When I wrote the first edition of this book, the hardest part of product management was the same as it had always been: deciding what to build, for whom, and why, and then getting a room full of smart, busy people to build it together. That is still the hardest part. What has changed is almost everything around it.

In the space of about two years, AI went from a feature on a few roadmaps to a working layer of the product team itself. By mid-2025, 88 percent of organizations told McKinsey they were using AI in at least one business function, up from 78 percent a year earlier, and 62 percent said they were at least experimenting with AI agents.[1] Product managers now have tools that draft requirements, summarize a week of customer calls before breakfast, watch dashboards overnight and flag what moved. Some of those tools are good. Some are confidently wrong. All of them change what a product manager's day looks like.

They do not change what a product manager is for. Someone still has to own the outcome. Someone still has to sit with a nurse at the end of a twelve-hour shift and understand why the software she was given makes her job harder. Someone still has to tell the CEO that the feature the biggest customer asked for is the wrong bet, and be right about it. AI makes the work faster. It does not make the judgment optional. If anything, the judgment matters more, because a team that can build twice as fast can also build the wrong thing twice as fast.

So this edition does three things the first did not.

It covers the numbers. The first edition promised chapters on profit and loss, unit economics and portfolio management, and did not deliver them. Chapters 6 and 7 do. A product manager who cannot read a P&L is negotiating with one hand tied behind their back.

It puts AI where it belongs: inside the work. Every chapter ends its main argument with a section called *How AI changes this*, and Chapter 17 is devoted to building AI products and leading teams that include AI agents. I have tried to be specific about what to hand to an agent, what to keep, and how to tell the difference.

It shows the work. Most chapters open with a scene from one fictional company, Brindlepath, followed through the whole book, and most include worked examples, templates and a one-page checklist. The figures are diagrams and charts you can use in your own planning, not decoration.

I have spent most of my career in product, much of it in healthcare, where the stakes are high, the buyers are many and the regulations are real. That shows in the examples. But the book is written for product managers in any kind of software. If you can ship a product that a hospital will buy, a clinician will use and a compliance officer will sign off on, you can ship almost anything.

Robert Merrell

---

1. McKinsey & Company, *The State of AI in 2025: Agents, Innovation, and Transformation*, survey of 1,993 participants conducted June 25 to July 29, 2025.

How to Use This Book

The book is organized in five parts that follow the life of a product, from the first hint of a problem to a business that scales.

- **Part 1, Foundations** (Chapters 1 to 3): what the job is now, how to find problems worth solving, and how to validate them before you spend real money.
- **Part 2, Strategy** (Chapters 4 to 7): vision and positioning, prioritization and roadmaps, the financial side of product decisions, and managing more than one product.
- **Part 3, Building** (Chapters 8 to 11): the product team, working with engineering and design, using data well, and building trust through security, privacy and compliance.
- **Part 4, Customers** (Chapters 12 to 14): designing with empathy, taking a product to market, and keeping and growing customers.
- **Part 5, Growth** (Chapters 15 to 18): partnerships, scaling, building AI products, and an integrated playbook that ties it all together.

You can read it front to back, or jump to the chapter that matches the problem on your desk. Each chapter stands on its own, and cross-references point you to related material.

Meet Brindlepath

Most chapters begin with a scene from Brindlepath, a fictional company that sells care-coordination software to health systems and large physician groups. Its product helps nurse navigators and care managers track patients as they move between the hospital, specialists, home health and primary care, the handoffs where things most often fall through the cracks.

Brindlepath is invented. So are its people: Maya Smith, who joins as Vice President of Product in Chapter 1; Dev Raman, the engineering lead; Lena Park, the head of design; and the customers, clinicians and executives they work with. Brindlepath, its people, its customers and every event in its story are fictional. They are not based on any real company, client, employer or person, and any resemblance to real organizations or people is coincidental. The situations are composites of challenges that are common across the software industry, written to illustrate the ideas in each chapter. Following one company through the whole book shows how the decisions connect: the problem you choose in Chapter 2 shapes the pricing you can charge in Chapter 6, which shapes the team you can afford in Chapter 8.

Features in every chapter

- **How AI changes this.** What AI tools and agents can take on in that part of the work, what they cannot, and the traps to avoid.
- **Figures.** Frameworks and charts you can reuse. Charts built on real-world data cite their source. Charts that use made-up numbers to teach a point are labelled *illustrative*.
- **Chapter checklist.** A one-page summary of the questions to ask and the work to do, for when you come back to the chapter months later with a real decision to make.

Templates for the most useful artifacts, including a problem statement, a product requirements document, a launch checklist and a one-page roadmap, are collected in the Appendix.

PART 1

Foundations

CHAPTER 1

The Product Leader's Job, Now

"There is nothing so useless as doing efficiently that which should not be done at all."

— Peter Drucker

Monday, 7:10 a.m.

Maya Smith's first Monday as Vice President of Product at Brindlepath started before she reached the office. On the train, she opened the overnight digest her team's research agent had assembled: forty-one support tickets summarized into six themes, two competitor release notes, a spike in failed logins at one hospital customer, and a draft agenda for the quarterly business review with Ambermoor Health, Brindlepath's largest account.

It was a good digest. It was also, she noticed, wrong in one important way. The agent had grouped eleven tickets under "notification fatigue" and recommended adding a setting to mute alerts. Maya read three of the tickets in full. The nurses were not tired of alerts. They were getting alerts for patients who had already been discharged to another team, because the handoff between the hospital's care managers and the outpatient navigators did not update ownership in Brindlepath. Muting the alerts would hide the symptom and keep the disease. The real problem was that nobody owned the patient for the forty-eight hours after discharge.

That distinction, between the request and the problem underneath it, is the job. The agent did in twenty minutes what used to take a product manager most of a Monday. It could not do the part that mattered: notice that the pattern meant something different from what it looked like, and decide what to do about it.

This chapter is about that part.

What a product leader is actually for

Ask ten companies what a product manager does and you will get eleven answers. In some, the product manager writes tickets and runs standups. In others, the product manager sets strategy for a business line with its own profit and loss. The title covers both.

Underneath the variety, the role has one constant: **the product leader owns the outcome.** Engineering owns how the product is built. Design owns how it works for the people using it. Sales owns the deal. Product owns whether the whole thing creates value for customers and for the business, and whether the team is working on the right problem in the first place.

Owning the outcome means answering four questions continuously, not once:

1. **Is it valuable?** Will the people it is for choose to use it, or buy it, over what they do today?
2. **Is it usable?** Can they figure out how to get that value without a training manual?
3. **Is it feasible?** Can we build and run it with the time, skills and technology we have?
4. **Is it viable?** Does it work for the business: can we sell it, support it, price it profitably and stay within the law?

These four risks will be familiar to anyone who has read Marty Cagan's work, and they remain the best short description of the job.[1] This edition adds a fifth question that used to sit quietly inside viability and now deserves its own line:

5. **Is it trustworthy?** Will customers, regulators and the public trust it with their data, their decisions and, in healthcare, their health? When a product

includes AI, does it behave reliably, explain itself where it needs to, and fail safely?

Trust used to be a compliance checkbox at the end of a project. In a world where software makes recommendations about patients and AI writes answers that look authoritative whether or not they are right, it belongs at the start.

How the role got here

It helps to know where the job came from, because many organizations are still running an older version of it.

The feature coordinator (roughly the 1990s and early 2000s). In the era of boxed software and long release cycles, product management grew out of marketing and project management. The product manager gathered requirements from customers and sales, wrote them into a large specification, and handed it to engineering. Success meant shipping the spec on time.

The product owner (the 2000s and 2010s). Agile development broke big specifications into small increments and gave each team a backlog. The product manager became the person who prioritized that backlog and accepted the work. Delivery got faster, but in many companies the role shrank to managing tickets. Teams shipped steadily and still built things nobody used.

The outcome owner (the late 2010s onward). SaaS changed the economics. When customers pay every month and can leave any month, shipping is not success; adoption, retention and expansion are. Product managers began to be measured on outcomes: activation rates, net revenue retention, time saved for the user. Discovery, the work of finding out what to build before building it, became a core part of the job rather than a phase at the start.

The orchestrator (now). The current shift is less about what the product manager is accountable for and more about how the work gets done. A product manager now directs a team that includes AI tools and agents alongside engineers, designers and analysts. Research synthesis, first drafts of requirements,

competitive monitoring, data pulls and test-case generation can all be delegated. The product manager's leverage has grown enormously. So has the cost of pointing that leverage in the wrong direction.

Each stage absorbed the one before it. A great product leader today still needs to write a clear requirement, still needs to run a tight backlog, and still needs to be measured on outcomes. The new layer sits on top.

The four pillars of product leadership

The first edition described three pillars of SaaS product leadership. Experience since then has added a fourth.



Figure 1.1. The four pillars of product leadership. Each pillar rests on the same foundation: owning the outcome.

Pillar 1: Market understanding

You cannot build for a market you do not understand. Market understanding means knowing who has the problem, how badly, what they do about it today, who else is trying to solve it, and what constraints shape the solution. In healthcare those constraints include regulation, reimbursement and clinical workflow; in fintech they include licensing and fraud; in education, procurement cycles that follow the school year. Every market has its own physics. Chapters 2 and 3 are about learning them.

Pillar 2: Financial insight

This is the pillar most product managers neglect, and the one that most separates a product manager from a product leader. Every product decision is also a financial decision: what it costs to build, what it costs to sell and support, how it changes what customers will pay and how long they stay. A product leader who can walk into a budget meeting with a clear view of unit economics gets resources. One who cannot gets opinions. Chapters 6 and 7 cover the numbers in detail.

Pillar 3: Customer-centered innovation

Innovation is not novelty. It is a better way to get a job done for a real person. Customer-centered innovation means staying close enough to users to see the problems they have stopped noticing, and having the discipline to measure whether what you shipped actually helped. In healthcare, that includes the patient who never logs into your software but whose care depends on it.

Pillar 4: Trust and responsibility

The new pillar covers security, privacy, compliance, accessibility and, increasingly, responsible AI. It is not a department's job that product hands off; it is a property of the product that product leadership owns. A breach, a biased model or an inaccessible workflow is a product failure, not just a technical one. Chapter 11 covers trust by design, and Chapter 17 covers the specific risks of AI features.

All four pillars rest on the same foundation: owning the outcome. A product leader who is strong in three pillars and weak in one will be found out by that one.

What great looks like

Over the years I have hired, managed and worked beside a few hundred product managers. The great ones share habits that have little to do with frameworks.

They can say what problem they are solving in one sentence, without jargon. If you cannot explain the problem to a smart outsider, the team does not understand it either.

They talk to customers every week, not every quarter. Not sales calls and not escalations, though those matter too. Conversations whose only purpose is to understand how someone works.

They separate the request from the need. A customer asking for a mute button for alerts is telling you something true about their pain and something unreliable about the cure. Maya's Monday morning is the job in miniature.

They make decisions and write them down. A decision log, with the reasoning and the evidence, is worth more than any roadmap slide. It lets the team move without waiting for the product manager, and it lets everyone learn when a decision turns out wrong.

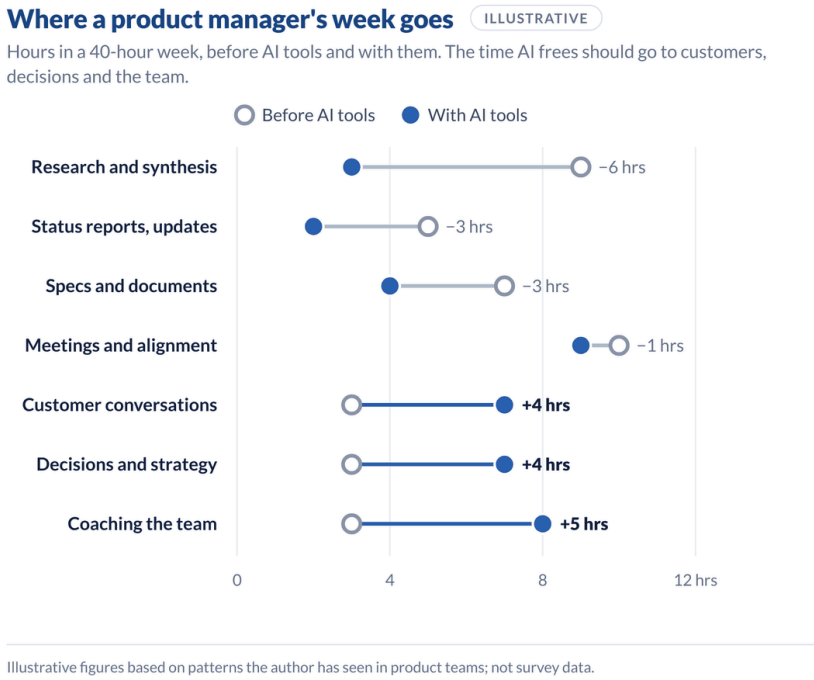
They say no, often and kindly. Every yes is a no to something else. Great product leaders protect the team's focus and explain the trade-off rather than hiding it.

They know their numbers. Not every metric in the dashboard: the three or four that tell them whether the product is working, and the one or two that tell them whether the business is.

They share credit and own failure. A product leader has authority over very little and influence over almost everything. Influence comes from trust, and trust comes from how you behave when things go wrong.

Where the week goes

One of the fastest ways to understand how the job is changing is to look at a product manager's calendar. Figure 1.2 compares a typical week for a product manager at a growing SaaS company before AI tools, and the same role with AI assistants and agents handling much of the preparation and synthesis. The numbers are illustrative, based on the pattern I see in teams I work with, not on a survey.



> Figure 1.2. Where a product manager's week goes, before and with AI tools (illustrative). AI takes over much of the synthesis and drafting. The time freed should go to customers, decisions and the team, not to more meetings.

Two things stand out. First, the time saved is real and large, mostly in research synthesis, status reporting and first drafts. Second, and more important, the time does not automatically flow to the work that matters. In teams that adopt AI tools without changing their habits, the freed hours fill with more meetings

and more documents. In teams that do it well, the hours go to customer conversations, sharper decisions and coaching. That reallocation is a leadership choice, not a side effect of the tools.

Brindlepath in practice: the decision log

One of the first habits Maya introduced was a decision log: a shared document where every significant product decision was recorded with five fields: the date, the decision, the options considered, the evidence, and what would make the team revisit it.

The first entry was the one from her Monday morning. *Decision: do not build a mute setting for notifications. Options: mute setting; change notification rules; fix ownership after discharge. Evidence: 11 tickets, three read in full, all caused by patients already discharged to another team. Revisit if: notification complaints continue after ownership is fixed.*

Within a few months the log had changed how the company argued. When the head of sales pushed again for the mute setting, Maya did not have to win the argument from memory; she pointed to the entry and asked whether the evidence had changed. It had not. When a decision did turn out wrong, such as an early choice to show discharges from all units by default, the log made it easy to see why it had seemed right at the time and what the team had missed. Nobody was blamed. The team simply learned faster.

The log also became the best onboarding tool the company had. New product managers read the last six months of entries in their first week and understood more about the product's history and reasoning than any slide deck could tell them. Dev Raman's engineers began reading it too, because it told them why they were building things, not just what. Two of them started adding their own entries for technical decisions that affected the product.

The decision log is not complicated, and it is not glamorous. It is one of the cheapest ways a product leader can make the team's judgment visible, shareable and improvable, which is, in the end, what owning the outcome requires.

Common mistakes

Acting as the team's boss. Product managers who issue instructions to engineers and designers get compliance, not ideas. The role carries accountability for the outcome, not authority over how others do their work. Lead through clarity, evidence and trust.

Becoming the team's secretary. The opposite failure: taking notes, chasing tickets and scheduling meetings while nobody owns the hard questions. If most of your week is coordination, you are doing a project manager's job, and nobody is doing yours.

Confusing busy with valuable. A full calendar and a long list of shipped features feel like progress. The question is whether anything changed for customers. Review your last month: which of your hours moved an outcome?

Ignoring the business. Product managers who talk only about users and never about revenue, margin or retention end up with roadmaps the leadership team does not trust. The four pillars need each other.

Treating trust as someone else's job. Security, privacy and accessibility failures land on the product, and on its leader, regardless of which department was supposed to catch them.

Letting tools set the agenda. New AI tools arrive every month. Adopting them without changing how you work just produces more output. Decide what you want more time for, then choose tools that create it.

Try this

Take one week of your calendar and sort every hour into four groups: customer understanding, decisions and strategy, team leadership, and coordination or reporting. Then ask three questions. Which group is largest? Which recurring tasks in the coordination group could an AI assistant draft for you? And what would you do with five extra hours a week if you had them? Write the answer down, and protect those hours on next week's calendar before anything else fills them.

How AI changes this

AI changes the product leader's job in three ways, and it is worth being precise about each.

It removes toil. Synthesizing interviews, summarizing support tickets, drafting requirements, writing release notes, pulling data, monitoring competitors, preparing meeting briefs: all of these can now be done in minutes by a well-instructed assistant. A product manager who still does them entirely by hand is choosing to be slower than their peers.

It raises the bar on judgment. Because the first draft is now cheap, the value moves to knowing whether the draft is right. Maya's research agent produced a plausible, well-written, wrong conclusion. The skill is not producing the summary; it is reading three tickets in full when the summary does not smell right. Product leaders need to become excellent editors of machine output: checking sources, spotting confident errors, and asking what the summary left out.

It changes what the team can attempt. A small team with good AI tooling can prototype, test and learn at a pace that used to require a much larger organization. That makes discovery cheaper and lowers the cost of being wrong early, which is good. It also makes it easier to drown in output, which is not. The constraint shifts from capacity to clarity: the team that knows exactly what problem it is solving will outrun the team with more tools.

What does not change: accountability, relationships and taste. An agent cannot be accountable for a product decision, cannot earn the trust of a skeptical chief medical officer, and cannot feel the difference between a product that is merely correct and one that people love. Those stay with you.

A practical rule I use with teams: **delegate the first draft, never the final call.** Let agents gather, sort, summarize and propose. Keep humans on decisions that affect customers, money, safety or people, and make sure a human has actually read the evidence behind each one.

Why SaaS, and why healthcare examples

Most of the principles in this book apply to any product. I focus on software sold as a service because the subscription model makes the product leader's job uniquely continuous. There is no launch-and-move-on. Every month, customers decide again whether the product is worth paying for, and every release can raise or lower that answer.

The examples lean on healthcare because that is where I have spent most of my career, and because healthcare is a demanding teacher. It has every hard problem in one place: many stakeholders with different goals, a buyer who is rarely the user, heavy regulation, entrenched legacy systems, and real consequences when software fails. The lessons travel. If your product serves banks, schools, factories or retailers, swap in your own regulations and stakeholders and the frameworks hold.

Chapter checklist

- [ ] Can you state, in one sentence, the outcome your product is accountable for?
- [ ] For your current top priority, have you answered all five questions: valuable, usable, feasible, viable and trustworthy?
- [ ] Which of the four pillars is your weakest? What will you do this quarter to strengthen it?
- [ ] When did you last talk to a customer with no agenda other than understanding their work?
- [ ] Do you keep a decision log that the team can read?
- [ ] Which recurring tasks in your week could an AI assistant draft for you? What will you do with the time?
- [ ] Where are you relying on AI output without reading the underlying evidence?

1. Marty Cagan, *Inspired: How to Create Tech Products Customers Love*, 2nd ed. (Wiley, 2017). Cagan describes four big risks: value, usability, feasibility and business viability.

CHAPTER 2

Finding Problems Worth Solving

“People don’t want to buy a quarter-inch drill. They want a quarter-inch hole!”

— Theodore Levitt, quoting Leo McGinneva

The scheduling problem that wasn’t

Brindlepath was founded to fix scheduling. Its founders had watched patients leave the hospital with a stack of follow-up appointments they could not keep: the cardiology visit set for a Tuesday when the patient’s daughter, the only driver in the family, worked; the physical therapy slot across town with no bus route. Readmissions followed. The founders built a clever tool that let care managers book follow-ups around a patient’s real-life constraints.

Hospitals liked the demo. Very few bought it.

When Maya joined, she spent her first three weeks shadowing care managers at two customer sites and one hospital that had said no. What she found was that scheduling was a symptom. The deeper problem was that after discharge, nobody clearly owned the patient. The inpatient care manager’s job ended at the door. The primary care office did not know the patient had been admitted. The specialist had a referral but no context. A well-scheduled appointment did not help if nobody noticed the patient had missed it.

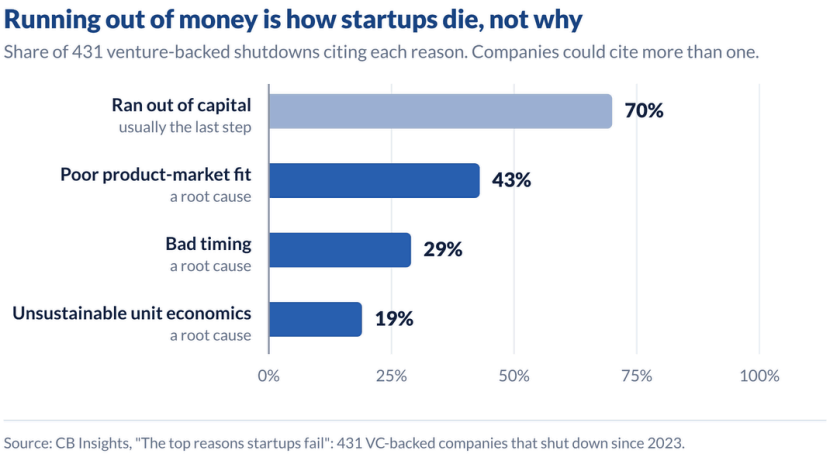
The care managers did not describe it that way. They described a dozen small frustrations: phone tag, faxes that went nowhere, discharge summaries that arrived after the follow-up visit, patients who “fell off the radar.” Maya’s job was to see the shape underneath the complaints. Once she did, Brindlepath’s product strategy changed from *better scheduling* to *clear ownership of every patient through every handoff*, and its sales conversations changed from feature demos to a problem every chief nursing officer recognized.

This chapter is about finding that shape, before you spend a year building the wrong thing.

Why products fail

Products rarely fail because the team could not build them. They fail because the team built something the market did not value enough, at a price or cost that did not work, at a time when nobody was ready.

CB Insights, which has studied startup failures for over a decade, analyzed 431 venture-backed companies that shut down from 2023 onward. Running out of capital was nearly universal, cited in 70 percent of cases, but CB Insights is careful to note that it is usually the final cause of death, not the root one.[1] Underneath it sat the familiar culprits: poor product-market fit, bad timing, and unit economics that never worked.



> Figure 2.1. Why venture-backed startups shut down. Running out of money is usually the last step, not the root cause. Source: CB Insights analysis of 431 VC-backed companies that shut down since 2023.

The lesson for product leaders inside established companies is the same. A new product line or major feature that fails rarely fails on engineering. It fails on the question this chapter is about: was this a problem worth solving, for people willing and able to pay for a solution?

Start with a problem statement

Every product effort should begin with a written problem statement. Not a solution statement dressed up as a problem ("Users need a dashboard") and not a vision ("Transform care coordination"). A problem statement names who is affected, what they are trying to do, what gets in the way, and what it costs them.

A useful template:

> **[Who]** need to **[goal or job]**, but **[obstacle]**, which leads to **[consequence, ideally measurable]**. Today they **[current workaround]**, which falls short because **[gap]**.

Here is Brindlepath's original problem statement, and the one Maya wrote after her field work:

Before: “Care managers need a better way to schedule post-discharge appointments.”

After: “Outpatient care navigators at health systems need to know, within 24 hours, which recently discharged patients are now theirs and what those patients need. But ownership after discharge is unclear, and information arrives by fax, phone and portal days late. As a result, a meaningful share of high-risk patients miss their first follow-up visit, which drives avoidable readmissions that the hospital is penalized for. Today navigators work from spreadsheets and daily phone calls with inpatient staff, which do not scale past a few dozen patients and break whenever someone is out.”

The second version is longer, and much more useful. It tells the team who the user is (the outpatient navigator, not the inpatient care manager), what success looks like (knowing within 24 hours), what the business case is (readmission penalties), and what the real competitor is (a spreadsheet and the telephone). Every one of those facts will shape the product.

A good problem statement passes four tests:

- **It contains no solution.** If it names a feature, rewrite it.
- **It names a specific person,** not “users” or “the market.”
- **It describes a consequence someone cares about,** in their terms: time, money, risk, outcomes.
- **It can be wrong.** A problem statement is a hypothesis. If no evidence could disprove it, it is too vague to be useful.

Jobs, not features

The most durable way to understand a problem is to ask what job the customer is hiring a product to do. The jobs-to-be-done idea, popularized by Clayton Christensen, holds that people do not buy products for their features; they “hire” them to make progress in a particular circumstance.[2] Levitt’s quarter-inch hole is the same idea, decades earlier.

Jobs have three layers:

- **Functional:** the practical task. *Know which discharged patients are mine today.*
- **Emotional:** how the person wants to feel. *Confident that nobody is falling through the cracks on my watch.*
- **Social:** how they want to be seen. *The navigator the physicians trust to have it handled.*

Products that serve only the functional job are easy to copy and easy to replace. Products that serve all three become part of how someone sees their own work. Brindlepath's navigators did not just want a list; they wanted to go home at night without the low-grade dread that a patient was in trouble and nobody knew.

When you interview customers, listen for the job behind the request. When someone asks for a feature, ask what they would do with it, and then what that would let them do, until you reach something they genuinely care about. That is usually the job.

Who has the problem, who pays, who decides

In consumer software, the person with the problem, the person who chooses the product and the person who pays are often the same. In business software they almost never are, and in healthcare the gap is extreme.

For a single Brindlepath deal, the cast typically includes:

- **The users:** outpatient navigators and care managers, who live in the product every day.
- **The clinical champion:** often a director of care management or a chief nursing officer, who feels the readmission problem and sponsors the purchase.
- **The economic buyer:** a chief financial officer or chief operating officer, who signs off on the money and wants a return.
- **The technical gatekeepers:** IT and information security, who must approve integration with the electronic health record and the handling of pa-

tient data.

- **The influencers:** physicians, quality leaders and compliance officers, who can slow or stop a deal without ever being in the room.
- **The beneficiary who never logs in:** the patient.

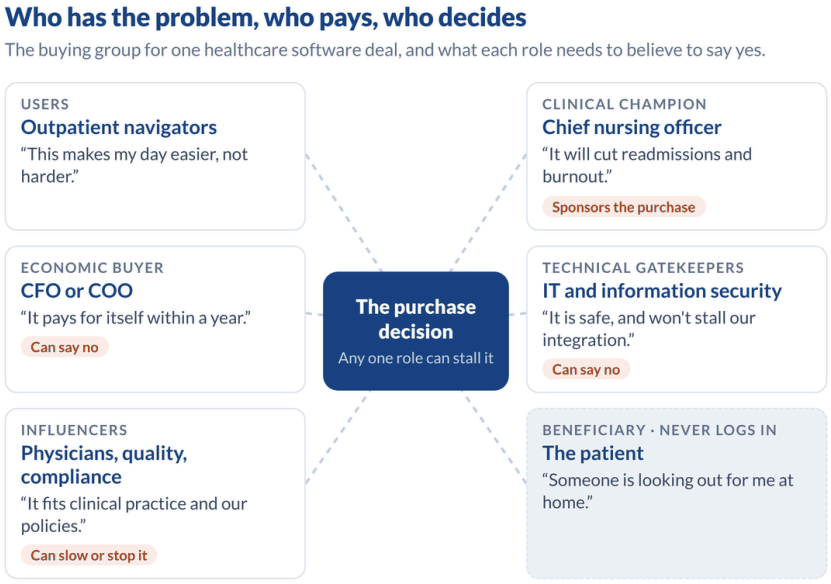


Figure 2.2. The buying group for a business-to-business healthcare product. Each role needs a different reason to say yes, and any one of them can say no.

Each of these people has a different version of the problem. The navigator cares about her day. The chief nursing officer cares about readmission rates and staff burnout. The CFO cares about penalties avoided and headcount not added. IT cares about integration effort and security risk. A product strategy that speaks only to the user will win enthusiastic pilots and lose the purchase.

Map your buying group explicitly. For each role, write down what they need to believe to say yes, what would make them say no, and what evidence would convince them. You will use this map again for positioning in Chapter 4, for pricing in Chapter 6 and for go-to-market in Chapter 13.

How big is the problem?

Not every real problem is worth solving. You need some way to compare problems before committing a team to one. I use three questions:

1. **How often does it happen?** A problem that occurs every day for every user beats one that occurs once a quarter.
2. **How much does it hurt?** Measure in the customer's currency: hours lost, money spent, risk carried, outcomes harmed.
3. **What are they doing about it now?** If people have built elaborate work-arounds, or are already paying for something imperfect, the problem is real and the budget exists. If they shrug, it may be real but not urgent.

Plotting candidate problems by frequency and severity gives you a quick, honest picture of where to focus. Figure 2.3 shows the exercise Maya ran with her team after the field work. The positions are from that exercise, using the team's own scoring; they are illustrative, not survey data.

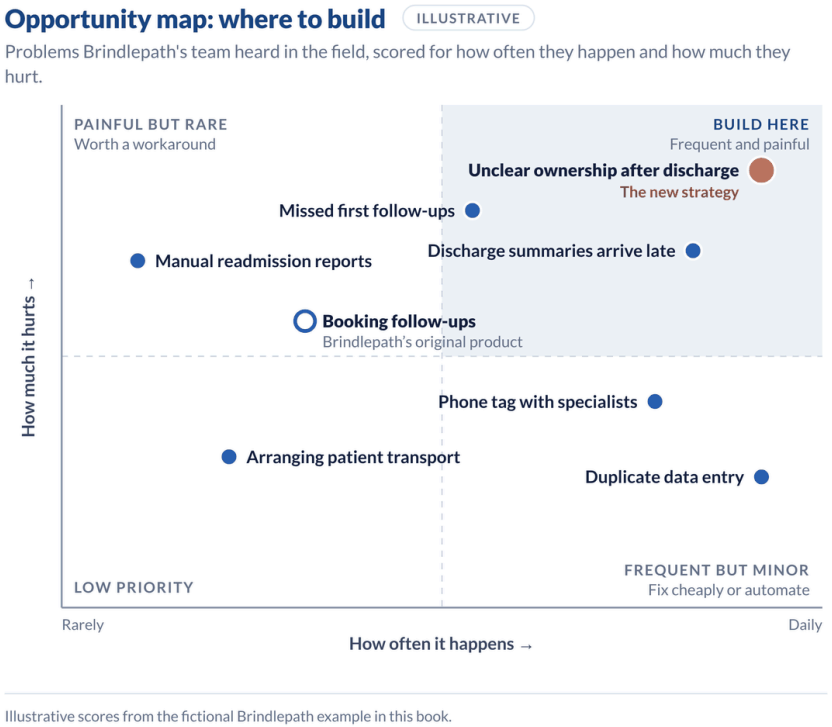


Figure 2.3. Opportunity map: problems scored by how often they occur and how much they hurt (illustrative). The top-right quadrant is where to build. The original product sat in the lower half.

The original scheduling product addressed a real but occasional pain. The ownership problem was frequent, severe and expensive. The map made the pivot obvious in a way no argument had.

Sizing the market

Once you have a problem worth solving, estimate how many people have it and what they would pay. Investors and executives will ask for three numbers:

- **Total addressable market (TAM):** everyone who has the problem, if you won every one of them.
- **Serviceable addressable market (SAM):** the part you can realistically reach with your product, channels and geography.

- **Serviceable obtainable market (SOM):** the share you can plausibly win in the next few years.

Build these numbers from the bottom up, not from analyst reports. Top-down estimates ("the care-coordination market is $X billion; if we get 1 percent…") are easy to produce and impossible to act on. A bottom-up estimate starts with countable units and a price.

For Brindlepath, the bottom-up version looked like this: there are a few hundred health systems in the United States with enough post-discharge volume and readmission exposure to care; each employs a number of outpatient navigators and care managers; Brindlepath prices per navigator seat with a platform fee. Multiplying the number of target systems by a realistic number of seats and a realistic price gives a market estimate the sales team can build a plan around and the finance team can check. The exact figures matter less than the method: every number in the chain is something you can verify by talking to customers.

Know the real competition

When teams list competitors, they usually list companies that sell similar software. That is the smallest part of the competition. The biggest competitor for most new products is the status quo: the spreadsheet, the phone call, the weekly meeting, the person who "just knows."

For Brindlepath, the competitive set included:

- **Direct competitors:** other care-coordination platforms.
- **The incumbent platform:** the electronic health record vendor, which offered basic care-management modules that were "good enough" for some buyers and already paid for.
- **The status quo:** spreadsheets, shared inboxes and daily huddle calls.
- **Doing nothing:** accepting the readmission penalties as a cost of doing business.

Each requires a different answer. Against direct competitors, you win on fit and outcomes. Against the incumbent platform, you win on depth and speed. Against the status quo, you win by making the cost of the workaround visible and the switch painless. Against doing nothing, you win only if the problem is urgent enough that someone's job depends on fixing it.

When you study competitors, study their customers more than their features. Read their reviews, talk to people who chose them and people who left them, and find out what job they were hired for and where they fall short.

Constraints are part of the problem

Every market has rules that shape what a solution can be. In healthcare, they include privacy law such as HIPAA in the United States, the need to integrate with electronic health records through standards like HL7 and FHIR, clinical safety expectations and, for some software, medical device regulation. Fintech has its licensing and anti-money-laundering rules; education has student-privacy laws; most enterprise software faces security reviews and procurement requirements.

Treat these constraints as part of the problem definition, not as obstacles discovered at the end. Ask early:

- What data will the product touch, and what rules govern it?
- What systems must it connect to, and how open are they?
- Who has to approve it before a customer can buy or deploy it?
- Could any part of it be regulated as more than ordinary software?

The answers change your cost, your timeline and sometimes your entire approach. A product that needs deep integration with five different electronic health records is a different business from one that works on a standalone basis. Better to know before you build. Chapter 11 covers trust and compliance in depth.

Brindlepath in practice: sizing the problem

When Maya presented the new problem statement to the board, the first question was the one every board asks: how big is this? She answered with a bottom-up estimate, built with Priya Nair, the CFO, and checked against conversations with a dozen health systems.

The chain of reasoning was simple. The team counted the health systems in Brindlepath's target regions with enough post-discharge volume and readmission penalties to feel the problem, and found a few hundred. Field interviews suggested that a typical system in that group employed somewhere between twenty and sixty outpatient navigators and care managers who would use the product. At Brindlepath's planned price of $1,800 per seat plus a platform fee, a typical customer would be worth roughly $70,000 to $150,000 a year. Multiplying the number of reachable systems by a conservative share and a typical contract value gave a serviceable market in the tens of millions of dollars of annual revenue, before any expansion into physician groups or health plans.

The number itself mattered less than the conversation it enabled. Because every link in the chain was something a board member could question and a salesperson could verify, the debate moved from "is this big enough?" to "which assumptions are weakest?" The weakest turned out to be the number of navigators per customer, which varied widely. So the team made it a discovery question for the next round of interviews, and refined the estimate a quarter later.

The same exercise exposed something uncomfortable about the old scheduling product. Fewer buyers felt that pain strongly, the budget for it was smaller, and the record vendor's module already did "good enough" scheduling for many. The bottom-up math, as much as the field work, made the case for the new strategy.

Common mistakes

Falling in love with a solution. Teams often start with an idea and then look for a problem to justify it. The tell is a problem statement that quietly describes the product. Brindlepath's original scheduling tool was a solution in search of the real problem.

Asking customers what they want. Customers are experts in their problems and unreliable designers of solutions. "Would you use this?" invites a polite yes. "Tell me about the last time this happened" invites the truth.

Talking only to the people who like you. Existing champions confirm what you believe. Prospects who said no, and customers who left, tell you what you are missing.

Ignoring the buyer. A product that delights users but cannot win over the economic buyer, IT and compliance will collect enthusiastic pilots and few contracts.

Sizing the market from the top down. "One percent of a ten-billion-dollar market" convinces nobody who has sold enterprise software. Build the estimate from countable customers and a realistic price.

Discovering constraints last. Regulation, integration and procurement requirements discovered after the product is built can double its cost or delay it by a year.

Try this

Write the problem statement for the most important thing your team is working on, using the template in this chapter. Then show it to three people: a customer, a colleague in sales or customer success, and an engineer. Ask each of them one question: "What is wrong with this?" Rewrite it after each conversation. If the statement changes substantially, you have learned something important before spending another week building. Finally, add the buying group: list each role, what they need to believe, and the evidence that would convince them.

How AI changes this

AI is genuinely useful in problem discovery, as long as you use it for breadth and keep humans on depth.

Use AI to scan wide. An assistant can read hundreds of support tickets, app-store reviews, sales-call transcripts, forum threads and competitor release notes, and cluster them into candidate problems. Work that once took weeks now takes an afternoon. Treat the clusters as leads, not conclusions.

Use AI to pressure-test your thinking. Ask a model to argue against your problem statement, list the stakeholders you have missed, or role-play a skeptical CFO. It will not always be right, but it is a fast, tireless sparring partner.

Use AI to draft the market model. Give it your bottom-up assumptions and ask it to build the spreadsheet and the sensitivity analysis. Then check every assumption yourself.

Do not use AI as a substitute for the field. A model trained on the internet knows what has been written about a problem, which is the average, often outdated view. The insight that changed Brindlepath's strategy came from standing behind a navigator at 4:30 p.m. and watching her call three offices to find out who owned a patient. No dataset contained that. The most valuable discovery still happens in person, and it is the part of the job that AI makes more important, not less, because everyone else now has the same desk research you do.

Watch for synthetic certainty. AI summaries of customer feedback tend to smooth over disagreement and produce tidy themes. Real problems are messy. When a cluster looks too neat, read the raw material.

Chapter checklist

- [ ] Have you written a problem statement that names a specific person, their goal, the obstacle, the consequence and the current workaround, with no solution in it?

- [ ] Can you describe the functional, emotional and social job your product does?
- [ ] Have you mapped the full buying group, and what each role needs to believe to say yes?
- [ ] Have you scored candidate problems by frequency and severity, and do you know what customers do about them today?
- [ ] Is your market estimate built from the bottom up, from numbers you could verify with customers?
- [ ] Have you listed the status quo and “do nothing” as competitors?
- [ ] Do you know the regulatory, data and integration constraints before you design a solution?
- [ ] Have you used AI to scan widely, and then gone into the field to see the problem for yourself?

---

1. CB Insights, “The top reasons startups fail,” analysis of 431 venture-backed companies that shut down since 2023. Other causes cited include poor product-market fit (43 percent), bad timing (29 percent) and unsustainable unit economics (19 percent). Companies can cite more than one reason.
2. Clayton M. Christensen, Taddy Hall, Karen Dillon and David S. Duncan, *Competing Against Luck: The Story of Innovation and Customer Choice* (Harper Business, 2016).

Chapter 3. Discovery and Validation, from Interviews to MVP

“There are no facts inside your building, so get outside.”

— Steve Blank

Twelve navigators and a spreadsheet

Maya had a problem worth solving. She did not yet have a product, and she knew better than to let the team start building one.

Instead, she and Lena Park, Brindlepath’s head of design, recruited twelve outpatient navigators from three health systems: two existing customers and the hospital that had said no. They did not show them a prototype. They asked each navigator to walk them through the last three patients who had “fallen off the radar,” what had happened, when they had found out, and what they had done.

The stories were remarkably consistent. The navigator usually found out late, from the patient, a family member or an emergency department visit. The information she needed existed somewhere: in the discharge summary, the inpatient care manager’s notes, the specialist’s referral. It just never arrived in one place, at the right time, with a name attached.

Then Dev Raman, the engineering lead, did something that made the engineers on his team uncomfortable. Rather than build anything, he set up a shared spreadsheet for one navigator team at a customer site, and for two weeks a Brindlepath analyst filled it in by hand every morning: newly discharged patients, who the inpatient contact was, what the discharge summary said, and a suggested owner. The navigators used it every day. By the second week, they were asking for it before 8 a.m. One navigator caught two patients who would otherwise have been missed.

Nobody at the customer knew there was no software behind it. Two weeks and one analyst's time had answered the most expensive question Brindlepath faced: if we deliver this, will they use it? Now the team could build with confidence.

Discovery is continuous, not a phase

In older ways of working, discovery happened once, at the start of a project, and produced a requirements document. Then the team built for months and found out whether they had been right.

Modern product teams treat discovery as a continuous practice that runs alongside delivery. Every week, the team learns something about its customers and tests at least one assumption. Teresa Torres calls this *continuous discovery* and offers a simple benchmark: a product team should be talking with customers at least weekly.[1]

Continuous discovery matters for three reasons:

- **Markets move.** What was true at the start of a project may not be true at the end.
- **Small bets beat big ones.** Testing an assumption in a week is cheaper than building on it for a quarter.
- **Teams that hear from customers make better small decisions.** Hundreds of tiny choices about wording, defaults and edge cases go better when the people making them have heard real users recently.

The rhythm does not need to be elaborate. A product trio of a product manager, a designer and an engineering lead, holding one or two customer conversations a week and reviewing what they learned, will outperform most formal research programs.

Interviewing well

Customer interviews are the most important discovery tool and the most commonly done badly. The most common mistake is asking people to predict their own future behavior or to evaluate your idea. People are polite and optimistic. They will tell you your idea is great and that they would definitely use it. Then they won't.

Rob Fitzpatrick's rule of thumb is to talk about their life, not your idea; ask about specifics in the past, not generics or opinions about the future; and talk less, listen more.[2] In practice:

Ask about the last time, not "usually." "Tell me about the last patient you lost track of after discharge" produces a real story with real details. "How do you usually handle discharges?" produces an idealized process that matches the policy manual.

Follow the workaround. When someone mentions a spreadsheet, a sticky note or a daily phone call, ask to see it. Workarounds are evidence of a problem worth solving and a blueprint for the solution.

Ask what it cost. Time, money, stress, consequences. If they cannot name a cost, the problem may not be urgent.

Ask what they have already tried. Someone who has never looked for a solution probably does not feel the pain strongly enough to buy one.

Avoid pitching. The moment you describe your product, the interview becomes a sales call and the data becomes polite.

Interview the people who said no. Lost deals and churned customers teach more than happy ones. Maya's most useful interviews were at the hospital that had rejected Brindlepath.

After each interview, capture the evidence while it is fresh: quotes, stories, workarounds and surprises, separately from your interpretation. Your interpretation will change; the evidence should not.

Map your assumptions

Every product idea rests on a stack of assumptions. Most teams never write them down, which means they never test them. A simple exercise changes that.

List every assumption that must be true for the idea to succeed, grouped by type:

- **Desirability:** *Navigators will check a daily list of newly assigned patients.*
- **Usability:** *They can confirm or reassign ownership in under a minute per patient.*
- **Feasibility:** *We can get discharge data from the customer's health record within a few hours of discharge.*
- **Viability:** *Health systems will pay per navigator seat, at a price that supports our margins.*
- **Trust and compliance:** *Customers' security teams will approve the data flow without a six-month review.*

Then rate each assumption on two scales: how important it is (if it is wrong, does the idea fail?) and how much evidence you already have. The assumptions that are important and unproven are the ones to test first. This sounds obvious, yet many teams spend their first months polishing the user interface, the part they are most confident about, while the riskiest assumption, often a feasibility or viability question, waits untested until late.

For Brindlepath, the riskiest assumption was not whether navigators wanted a list. The field work had largely answered that. It was whether discharge data could flow from five different health record configurations quickly and reliably. So the team's first technical spike was an integration test with one customer's health record, not a user interface.

The evidence ladder

Not all evidence is equal. An enthusiastic comment in an interview is evidence; a customer changing their behavior is stronger evidence; a customer paying money is stronger still. Figure 3.1 arranges common validation methods by the strength of evidence they produce and the cost of running them.

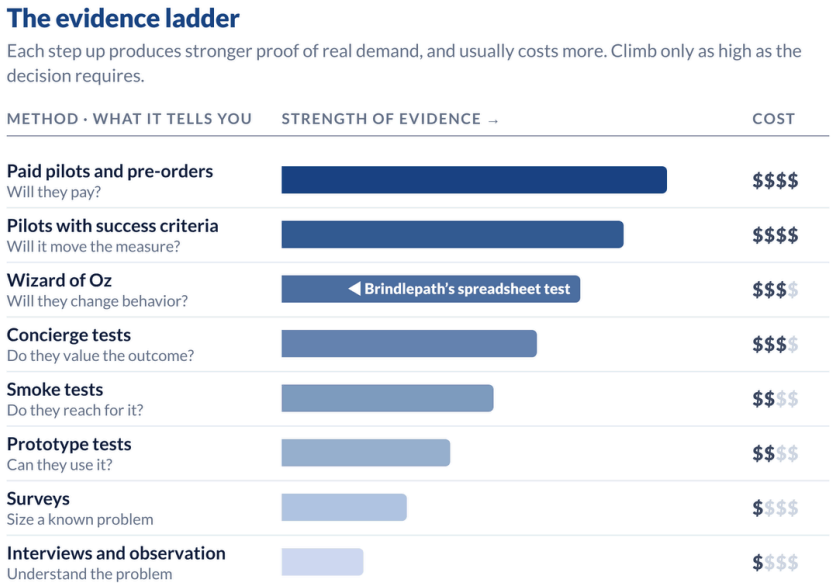


Figure 3.1. The evidence ladder. Each step up produces stronger evidence of real demand, and usually costs more. Climb only as high as the decision requires.

The principle: **match the strength of evidence to the size of the decision.** A small feature change may need nothing more than a few interviews and a prototype test. A new product line with a dedicated team deserves evidence from the top of the ladder: commitments, pilots with success criteria, or paid pre-orders.

The experiments in detail

Interviews and observation. The base of the ladder. Cheap, fast and essential for understanding the problem; weak as evidence that people will adopt a specific solution.

Surveys. Useful for sizing a problem you already understand, measuring how widespread a behavior is or prioritizing among known options. Poor for discovering problems, and poor at predicting behavior.

Prototype tests. Put a clickable prototype in front of users and give them a task. You learn whether they understand the design and can use it. You do not learn whether they will use it every day.

Smoke tests. Describe a product or feature as if it exists, on a landing page or in an in-app prompt, and measure how many people try to use it or sign up. Common in consumer products; in business software, a "request early access" button for existing customers works well.

Concierge tests. Deliver the outcome by hand, openly, to a small number of customers. You learn exactly what the service needs to do, and whether customers value it enough to keep using it.

Wizard of Oz tests. Deliver the outcome by hand behind something that looks like a product, as Brindlepath did with its spreadsheet. You learn whether customers will change their behavior when the experience feels real.

Letters of intent and pilots with success criteria. A customer formally commits time, data or money to a trial with agreed measures of success and a decision date.

Paid pilots and pre-orders. The strongest evidence short of a full sale: someone gives you money for something that is not finished.

Minimum viable product, done right

The term *minimum viable product* is one of the most misused in the industry. It does not mean "the first release, with fewer features." It means the smallest thing you can build that delivers real value to real users and lets you learn whether your core assumption is true.

Two words carry the weight. **Minimum** means you cut everything that is not needed to test the core value. **Viable** means what remains must genuinely work for the user; a broken or embarrassing product teaches you nothing, be-

cause users reject it for the wrong reasons.

There is good evidence that most teams build far more than they need. In a 2019 analysis of usage across 615 of its software customers, the product-analytics company Pendo found that 80 percent of features in the average product were rarely or never used, and that about 12 percent of features generated 80 percent of daily usage.[3] The data comes from one vendor's customers and is several years old, but it matches what most product leaders see in their own analytics.

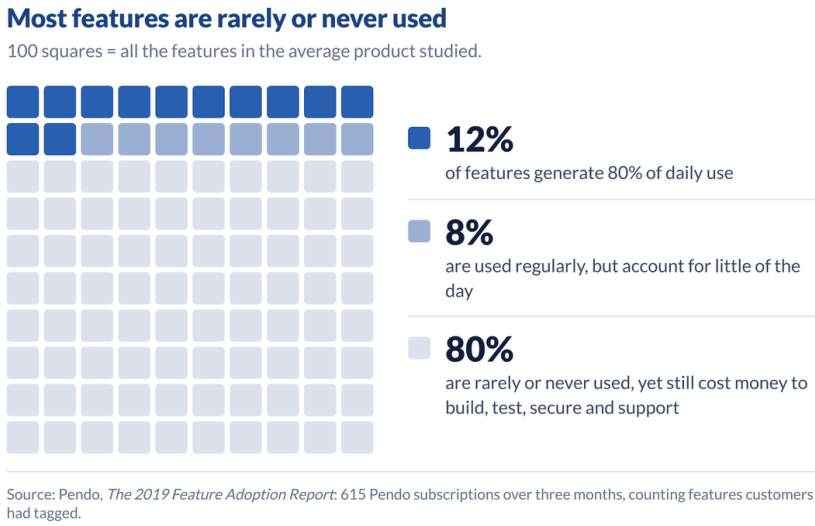


Figure 3.2. Most features are rarely or never used. In Pendo's 2019 analysis, about 12 percent of features drove 80 percent of daily use. Source: Pendo, 2019 Feature Adoption Report.

Every feature that does not earn its place costs money to build, test, document, support, secure and maintain, for as long as the product exists. The cheapest feature is the one you were disciplined enough not to build.

Brindlepath's MVP

After the spreadsheet test, Maya's team defined the minimum viable product in one sentence: *Every morning, each outpatient navigator sees the patients discharged in the past 24 hours who are now theirs, with the discharge sum-*

mary and the inpatient contact attached, and can accept or reassign each one.

What they deliberately left out of the first release:

- Scheduling, the original product's core feature. Navigators could keep using the existing tool.
- Analytics dashboards for managers.
- Patient-facing messaging.
- Support for every health record vendor. They started with the two configurations used by their pilot customers.

What they insisted on including, even in the MVP:

- Role-based access controls and full audit logging, because patient data was involved and the customer's security team would not approve anything less.
- A way to reassign a patient in one step, because the field work showed ownership disputes were common.
- A measure of the outcome: the share of high-risk discharged patients with a confirmed owner within 24 hours.

The list of exclusions is as much a part of the MVP as the list of features. Write it down, share it, and defend it.

Pilots that end in a decision

In business-to-business software, and especially in healthcare, validation often takes the form of a pilot. Pilots are valuable. They are also where many promising products go to die, in what people in the industry sometimes call "pilot purgatory": a trial that is never declared a success or a failure and simply fades.

A pilot that ends in a decision has five things agreed in writing before it starts:

1. **The problem and the measure.** What outcome are we trying to change, and how will we measure it? For Brindlepath: the share of high-risk dis-

charges with a confirmed owner within 24 hours, and missed first follow-ups.

2. **The baseline.** What is the measure today? If nobody knows, measure it before the pilot begins.
3. **The success threshold.** What result would justify a purchase? Agree on a number, not on "a good result."
4. **The scope and duration.** Which teams, how many users, how long. Long enough to see the outcome; short enough to keep momentum. For most workflow products, 60 to 90 days is enough.
5. **The decision and the decision-maker.** Who decides, on what date, and what happens next if the threshold is met: a signed contract, a wider rollout, a budget line.

Get the economic buyer, not just the enthusiastic user, to agree to these terms. A pilot sponsored only by the people who love the product will produce a happy report and no purchase order. Chapter 13 returns to pilots as part of go-to-market.

Brindlepath in practice: the pilot agreement

The spreadsheet test answered the desirability question. The next step was a pilot that could turn into a contract. Maya, Lena and the sales lead wrote a one-page pilot agreement with the second health system, using the template in the Appendix.

The measure was the share of high-risk discharges with a confirmed owner within 24 hours, along with missed first follow-up visits. The baseline took two weeks to establish, because nobody at the health system had measured it before: about 45 percent of high-risk patients had a clear owner within a day. The success threshold was 75 percent, sustained for the last four weeks of a 90-day pilot. The scope was one hospital and its outpatient navigator team of fourteen people. The health system paid a modest pilot fee and committed a navigator manager's time each week. The decision-maker was the chief nurs-

ing officer, with the CFO's agreement, and the decision date was written into the agreement, along with the price and timeline for a rollout to the system's other four hospitals if the threshold was met.

Three things went wrong during the pilot, which is normal. The discharge feed missed patients from one unit for a week. Two navigators were on leave in the second month. And a change to the hospital's risk-scoring rules briefly flooded the queue with low-risk patients. Because the pilot had a clear measure and a weekly review, each problem was caught and fixed within days.

By day 90, the measure stood at 82 percent. The decision meeting took twenty minutes, because nothing had to be negotiated: the terms had been agreed at the start. The rollout to four more hospitals began the following month.

Common mistakes

Running discovery as a phase. Teams that do a burst of research at the start of a project and then stop are surprised by what they learn at launch. Discovery is a weekly habit.

Pitching instead of listening. The moment you describe your idea in an interview, the data becomes polite. Save demos for later, and keep early conversations about the customer's world.

Testing the safe assumptions. It is tempting to test what you are already confident about, such as the interface, while the riskiest assumption, often feasibility or willingness to pay, waits. Test the assumption that would kill the idea first.

Building an MVP that is all minimum and no viable. A product so thin that it does not solve the problem teaches you nothing, because users reject it for the wrong reasons.

Treating the MVP as the finished product. The opposite error: launching the first slice and moving on. An MVP is the beginning of learning, not the end of the project.

Running pilots without a decision. A pilot with no agreed measure, threshold or decision-maker drifts into “pilot purgatory,” consuming time and goodwill on both sides.

Try this

List the five assumptions your current idea depends on most, one each for desirability, usability, feasibility, viability and trust. Score each from one to five for how important it is and how much evidence you have. Pick the assumption with the highest importance and the least evidence, and design the cheapest experiment that could disprove it within two weeks. Use the evidence ladder in Figure 3.1 to choose the method. Then schedule the experiment, and decide in advance what result would change your plan.

How AI changes this

Discovery is one of the areas where AI helps most, and also where it can do the most damage if used carelessly.

Synthesis is now fast. Interview transcripts, support tickets and sales-call recordings can be transcribed, tagged and clustered by an AI assistant within hours of collection. A product trio that used to spend a week synthesizing a research round can now spend that week running the next one.

Prototypes are now cheap. AI coding and design tools can turn a sketch into a working prototype in an afternoon. That moves prototype testing down the cost axis of the evidence ladder and makes it reasonable to test several directions instead of one.

Humans talk; agents synthesize. The order matters. Use people for the conversations and the sense-making, and AI for the transcription, tagging, first-pass clustering and drafting. Figure 3.3 shows the division of labor that works in the teams I advise.

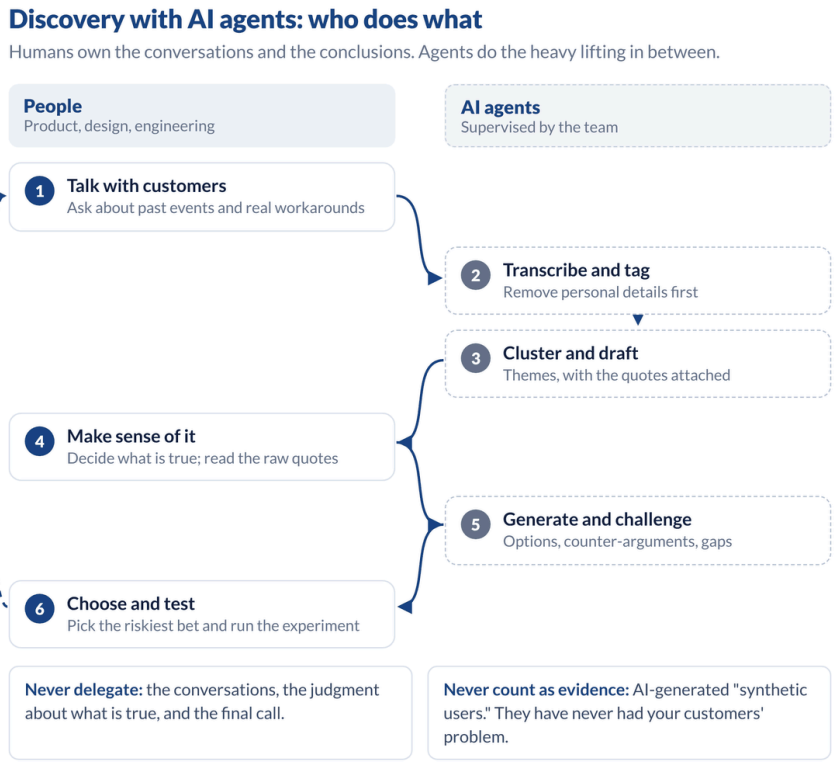


Figure 3.3. Discovery with AI agents: humans own the conversations and the conclusions; agents do the heavy lifting in between.

Synthetic users are not users. Some tools now offer AI-generated “personas” that answer interview questions. They can help you rehearse an interview guide or anticipate obvious objections. They cannot tell you anything true about your actual customers’ behavior, because they have never had your customers’ problem. Never count a synthetic interview as evidence.

Beware the smooth summary. AI is good at producing coherent themes from messy data, which is exactly the risk. Real customer evidence is contradictory and specific. Keep the raw quotes attached to every theme, and read a sample of the source material yourself. The navigator who caught two patients in the spreadsheet test is worth more than any summary of her.

Keep the patient's privacy in mind. In healthcare and other regulated settings, interview recordings and support tickets may contain personal or health information. Before you put them into any AI tool, confirm that the tool is approved for that data, and remove identifying details where you can. Chapter 11 covers this in depth.

Chapter checklist

- [ ] Does your product team talk with customers every week?
- [ ] Do your interviews ask about specific past events, not opinions about your idea?
- [ ] Have you interviewed people who said no, or who left?
- [ ] Have you listed your assumptions across desirability, usability, feasibility, viability and trust, and tested the riskiest first?
- [ ] Is the strength of your evidence matched to the size of the decision?
- [ ] Can you state your MVP in one sentence, with a written list of what it leaves out?
- [ ] Does every pilot have a measure, a baseline, a success threshold, a duration and a named decision-maker, agreed in writing?
- [ ] Are AI tools doing your synthesis while humans do the conversations and the conclusions?
- [ ] Have you confirmed that any AI tool touching customer data is approved for it?

---

1. Teresa Torres, *Continuous Discovery Habits: Discover Products that Create Customer Value and Business Value* (Product Talk, 2021).
2. Rob Fitzpatrick, *The Mom Test: How to Talk to Customers and Learn If Your Business Is a Good Idea When Everyone Is Lying to You* (2013).
3. Pendo, *The 2019 Feature Adoption Report*. The analysis covered 615 Pendo subscriptions over three months and counted features that customers had tagged in the product.

PART 2

Strategy

Chapter 4. Vision, Strategy and Positioning

“The essence of strategy is choosing what not to do.”

— Michael Porter

Three visions in one room

Six weeks after Maya Smith joined Brindlepath, the company held its annual planning offsite. By lunch on the first day, it was clear the leadership team did not share a picture of the future.

The CEO wanted Brindlepath to become “the operating system for care coordination,” a platform that every team in a health system would use. The head of sales wanted a product that could win the forty open opportunities in the pipeline, most of which were asking for scheduling improvements and a patient app. The chief medical officer, who had joined from a large health system, wanted the company to focus narrowly on high-risk patients after discharge, where the clinical evidence was strongest and the readmission penalties gave buyers a reason to act.

All three were reasonable. All three were, in their own way, strategies. And because nobody had chosen among them, the product team had been trying to serve all three at once, which is to say, none of them well.

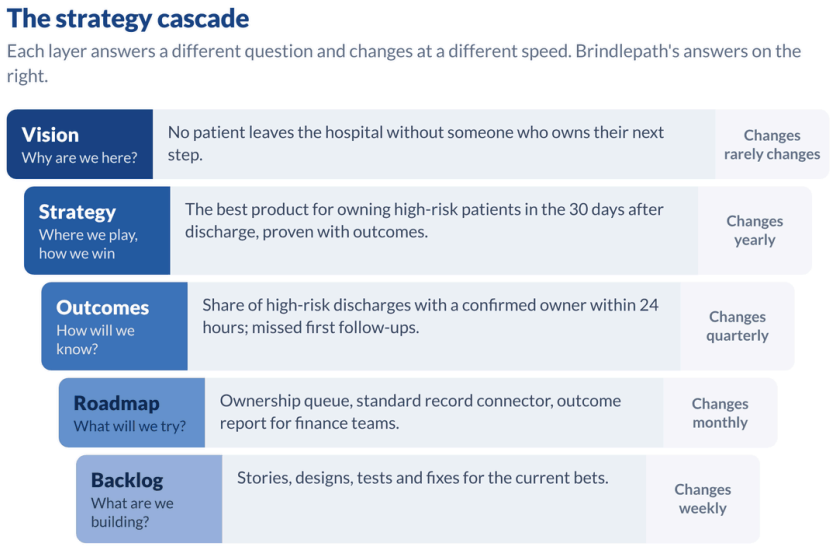
On the second morning, Maya put a single page on the screen. It did not contain a roadmap. It contained a diagnosis of the market, one choice about where Brindlepath would play, one claim about how it would win, and a short list of

things the company would stop doing. The argument that followed was the most productive two hours of the offsite.

This chapter is about writing that page, and about the work that makes it true.

Vision, strategy, roadmap: getting the words straight

Few words in business are used as loosely as *vision* and *strategy*. Teams waste hours arguing past each other because one person's strategy is another's roadmap. A simple cascade helps.



> Figure 4.1. The strategy cascade. Each layer answers a different question and changes at a different speed. Most teams skip the middle and go straight from vision to backlog.

- **Vision** describes the future you are trying to create for customers, usually three to ten years out. It answers *why are we here?* It should change rarely.
- **Strategy** is the set of choices about how you will get there: which customers and problems you will focus on, and how you will win against the alternatives. It answers *where will we play and how will we win?* It is reviewed every year or so, and changes when the market does.

- **Outcomes** are the measurable results that tell you the strategy is working: adoption, retention, revenue, the customer's own results. They answer *how will we know?* They are set quarterly or half-yearly.
- **Roadmap** is the sequence of bets the team will make to move those outcomes. It answers *what will we try, in what order?* It changes as you learn.
- **Backlog** is the detailed work to deliver the current bets. It changes every week.

The common failure is to skip the middle: a lofty vision on a slide, a detailed backlog in a ticketing tool, and nothing in between that explains why this backlog serves that vision. When the middle is missing, every stakeholder fills it with their own strategy, which is exactly what had happened at Brindlepath.

Writing a vision people can use

A useful product vision has four properties.

It describes the customer's world, not your product. "Brindlepath is the leading care-coordination platform" is about you. "No patient leaves the hospital without someone who owns their next step" is about the world you want to create. The second is far more useful, because it lets people judge any idea against a simple question: does this help make that true?

It is ambitious but believable. A vision nobody believes is decoration. A vision that is already true is a status report.

It is short enough to remember. If people cannot repeat it after hearing it twice, it will not guide their decisions.

It is concrete enough to say no. "Improve healthcare" excludes nothing. "Every patient has an owner through every handoff" excludes a great deal, including, as Brindlepath discovered, building a better scheduling tool for its own sake.

A good test: show the vision to a new engineer and ask them to name two things it rules out. If they cannot, it is too vague.

Strategy is a set of choices

The best short definition of strategy I know comes from Richard Rumelt, who argues that good strategy has a simple core: a **diagnosis** of the challenge, a **guiding policy** for dealing with it, and a set of **coherent actions** that carry out the policy.[1] Bad strategy, by contrast, is a list of goals, a pile of fluffy language, or a refusal to choose.

Applied to product, the three parts look like this:

Diagnosis. What is really going on in the market, and what is the hardest problem we face? Brindlepath's diagnosis: health systems are penalized for readmissions but have no reliable way to make sure someone owns each patient after discharge. Many vendors sell care-management modules, but few solve the handoff between inpatient and outpatient teams, because it requires integration with records, clear rules about ownership, and workflows that fit outpatient navigators. Brindlepath's original product had spread itself across scheduling, messaging and reporting, and was not decisively better at any of them.

Guiding policy. What approach will we take? Brindlepath's: become the best product in the market for owning high-risk patients in the 30 days after discharge, and prove it with outcomes, starting with health systems that face meaningful readmission penalties.

Coherent actions. What will we do, and stop doing, that follows from the policy? Invest in record integration and ownership workflows; build outcome reporting that a chief nursing officer can take to the CFO; stop new investment in the standalone scheduling product; decline deals that need a patient app as a condition of purchase; train the sales team to sell a problem, not a feature list.

Notice that the actions include things the company will stop doing. Michael Porter's line at the top of this chapter is the most-quoted sentence in strategy for a reason. A strategy that does not make anyone unhappy has probably not chosen anything.

Where to play, how to win

A complementary lens comes from A.G. Lafley and Roger Martin, who frame strategy as an integrated set of answers to a few questions, the most important being **where to play** and **how to win**.[2]

Where to play covers which customers, which problems, which channels and which geographies. *How to win* covers the advantage that will make those customers choose you: a better outcome, a lower cost, a superior experience, a unique capability or a trusted brand.

The two answers must fit each other. Brindlepath could not win on price against the electronic health record vendor's bundled module, and it could not win on breadth against large platforms. It could win on depth and outcomes in one specific, painful workflow. That determined where it should play: health systems where that workflow was a priority, not every organization that might buy care-management software.

Positioning: how customers should understand you

Strategy is what you decide. Positioning is how you help the market understand that decision. Even an excellent product fails if customers file it in the wrong mental drawer.

April Dunford's framework is the most practical I have used. She defines positioning through five components that need to fit together:[3]

1. **Competitive alternatives:** what customers would do if your product did not exist. Often the status quo.
2. **Unique attributes:** the capabilities you have that the alternatives lack.
3. **Value:** the benefit those attributes enable for customers, in their terms.
4. **Best-fit customers:** the characteristics of the customers who care most about that value.
5. **Market category:** the context you choose to present yourself in, which tells customers what to compare you with.

The order matters. Most teams start with the category and work backward. Dunford recommends starting with the alternatives, because your uniqueness only exists in comparison to something.



> Figure 4.2. Brindlepath's positioning, built from the alternatives outward. The category comes last, because it has to frame everything before it.

Brindlepath's old positioning put it in the category "care-management software," where buyers compared it, feature by feature, with large platforms and the record vendor's module, and usually found it smaller. The new positioning put it in a narrower category, "post-discharge care transitions," where the alternatives were spreadsheets, phone calls and modules that were not built for the handoff. In that frame, Brindlepath was not the smaller option; it was the specialist.

Positioning is not a tagline. It is the shared understanding that sits behind every tagline, sales deck, website page and product decision. Write it down, test it with customers and prospects, and expect it to change when your market does.

Communicating the vision and strategy

A strategy that lives in the head of the product leader is not a strategy. It is a hunch. The work of making it shared is less glamorous than writing it and far more important.

Write it on one page. If the strategy does not fit on a page, it is not yet clear. The page should contain the vision, the diagnosis, the guiding policy, the three to five most important actions, the outcomes you will measure, and what you will stop doing.

Tell it as a story. People remember stories, not bullet points. “A navigator at one of our customers found two patients last week who would have been missed” carries more weight than “improve post-discharge ownership rates.”

Repeat it until you are sick of it. By the time the product leader is bored of the strategy, most of the company has heard it once. Say it in all-hands meetings, planning sessions, customer calls and onboarding.

Connect every roadmap item back to it. When someone asks why the team is building something, the answer should trace back to the page in one or two steps.

Invite challenge, then decide. A strategy that has survived argument is stronger than one that was handed down. But a strategy that is re-argued every week is not a strategy. Hold a real debate, make the decision, write it down, and revisit it on a schedule rather than on demand.

The “not doing” list

The most powerful page in Maya’s offsite presentation was the last one: what Brindlepath would stop doing. It listed the standalone scheduling roadmap, the patient app, custom reporting for individual customers and two integrations requested by a single prospect.

Each item had a sponsor in the room. Each cut hurt. But the list did something no amount of prioritization could: it freed about a third of the engineering capacity for the work that mattered, and it told the sales team, clearly, which

deals to walk away from.

Write your own not-doing list. Share it as widely as the strategy itself. It is the clearest evidence that you have a strategy at all.

Brindlepath in practice: the one-page strategy

Here is the page Maya put on the screen at the offsite, lightly edited.

Vision. No patient leaves the hospital without someone who owns their next step.

Diagnosis. Health systems are penalized for readmissions but lack a reliable way to make sure someone owns each high-risk patient after discharge. Care-management modules exist, but few solve the handoff between inpatient and outpatient teams. We have spread ourselves across scheduling, messaging and reporting, and are not decisively better at any of them.

Where we play. Health systems with meaningful readmission penalties and dedicated outpatient navigator teams, starting in our current regions.

How we win. Depth in one workflow: discharge data within hours, clear ownership, one-step reassignment and outcome reporting, proven with customers' own numbers.

Guiding policy. Become the best product for owning high-risk patients in the 30 days after discharge, and prove it with outcomes.

Actions. Build the ownership queue and the standard record connector. Build outcome reporting for finance teams. Retrain sales to sell the problem. Move existing customers off custom integrations.

Outcomes. High-risk patients with a confirmed owner within 24 hours; renewal rate; time to first value.

Not doing. New investment in standalone scheduling; a patient app as a condition of sale; custom reports for individual customers; integrations requested by a single prospect.

The page was revised twice in the following year, both times on schedule. The not-doing list grew by one item and lost one: the patient-messaging need returned, but through an acquisition rather than a build, as Chapter 7 describes.

Common mistakes

Mistaking goals for strategy. "Grow revenue 40 percent" is a goal. A strategy explains how, given the market's realities, you will achieve it, and what you will not do along the way.

Writing a vision about yourself. Visions that describe the company ("the leading platform") inspire nobody and guide no decisions. Describe the customer's world.

Refusing to choose. A strategy that tries to serve every segment, match every competitor and say yes to every request is not a strategy. It is a list of hopes.

Positioning against the wrong alternative. Teams compare themselves to the competitor they admire, while customers compare them to the spreadsheet they already have. Start from what customers would really do without you.

Strategy by slide. A strategy presented once at an offsite and never repeated does not exist for most of the company. Repetition is not redundancy; it is how strategy becomes real.

Changing strategy every quarter. Constant change looks agile and produces teams that never finish anything. Review strategy on a schedule, and change it when evidence demands it.

Try this

Write your product's strategy on a single page, using the template in the Appendix: vision, diagnosis, where you play, how you win, guiding policy, three to five actions, outcomes and a not-doing list. Then run two tests. First, give it to a new engineer and ask them to name two ideas it rules out. Second,

ask a sales leader to describe, from the page alone, the customer you should walk away from. If either test fails, the strategy is not yet clear enough to guide decisions. Rewrite it until both pass.

How AI changes this

AI is reshaping product strategy in three ways that every product leader needs to think through.

Features commoditize faster. When a capability can be built in days with a foundation model, a feature on its own is rarely a lasting advantage. Competitors, including the large platforms, can match a clever summarization or drafting feature within a quarter. Durable advantage moves to the things that are hard to copy: deep integration into workflows, proprietary data and feedback loops, trusted relationships, distribution, and the accumulated understanding of a specific customer's problem. When you write your "how to win," be honest about which of these you actually have.

New competitive alternatives appear. For many products, a general-purpose AI assistant is now a competitive alternative. A navigator might paste a discharge summary into a chatbot instead of buying specialized software. Your positioning has to explain why your product is better than that, and in regulated settings the answer often includes trust: privacy, accuracy, auditability and integration that a general tool cannot offer.

AI is a great sparring partner for strategy. Ask a model to attack your diagnosis, list the strategies a competitor might pursue, or argue for the opposite choice. Ask it to rewrite your vision in plain language, or to find the actions in your plan that do not fit your guiding policy. It will not make the choice for you, but it will help you see where your thinking is thin.

A warning: AI also makes it easy to produce strategy documents that sound impressive and choose nothing. Fluent prose about "leveraging synergies across the care continuum" is now free. A real strategy still requires a person willing to say no.

Chapter checklist

- [ ] Can you state your product vision in one sentence that describes the customer's world, not your product?
- [ ] Does your strategy contain a diagnosis, a guiding policy and coherent actions?
- [ ] Have you answered where you will play and how you will win, and do those answers fit each other?
- [ ] Have you written your positioning starting from the competitive alternatives, including the status quo and general AI tools?
- [ ] Does your strategy fit on one page, and has the whole company heard it more than once?
- [ ] Can every roadmap item be traced back to the strategy in one or two steps?
- [ ] Do you have a written not-doing list, and have you shared it?
- [ ] Is your claimed advantage something competitors with the same AI tools cannot copy in a quarter?

---

1. Richard Rumelt, *Good Strategy Bad Strategy: The Difference and Why It Matters* (Crown Business, 2011).
2. A.G. Lafley and Roger L. Martin, *Playing to Win: How Strategy Really Works* (Harvard Business Review Press, 2013).
3. April Dunford, *Obviously Awesome: How to Nail Product Positioning So Customers Get It, Buy It, Love It* (Ambient Press, 2019).

Chapter 5. Prioritization and Outcome Roadmaps

“If everything is important, then nothing is.”

— Patrick Lencioni

Two hundred and twelve ideas

The week after the offsite, Maya asked for an export of everything that had ever been requested of the product. The list had 212 items. Some were bugs. Some were feature requests from a single customer, logged three years earlier by a salesperson who had since left. Some were genuinely important. Eleven were labelled “top priority.”

The list was not a backlog. It was an archaeological record of every conversation the company had ever had. And it had been driving the roadmap, because the loudest recent request usually won.

Maya did three things in her first month. She archived every item older than a year that nobody could connect to a current customer problem; 130 items disappeared, and in the following year, nobody asked for any of them back. She grouped what remained under the outcomes in the new strategy. And she replaced the old roadmap, a list of features with quarter-end dates, with one organized around problems and outcomes, with dates only where a commitment was genuinely required.

This chapter covers how to choose what to work on, and how to show those choices in a way that builds trust rather than false certainty.

Output versus outcome

The most important shift in modern product management is from output to outcome. **Output** is what the team ships: features, releases, documents. **Outcome** is the change in behavior or results that the output was meant to produce: more navigators confirming ownership within 24 hours, fewer missed follow-ups, higher renewal rates.

Output is easy to measure and easy to celebrate. Outcomes are what customers and the business actually care about. A team can ship everything on its roadmap and change nothing that matters. Josh Seiden put it simply: an outcome is a change in human behavior that drives business results.[1]

Working on outcomes changes how a team behaves:

- It starts each piece of work with a measurable goal, not a feature description.
- It treats the first release as a test, not a finish line.
- It stops or changes direction when the measure does not move, instead of shipping version two of something that did not work.
- It talks about results in reviews, not just about what was delivered.

Outcomes are harder to commit to, because you cannot guarantee that a customer will change their behavior. That discomfort is honest. A roadmap full of committed features is promising something the team does not control: that those features will matter.

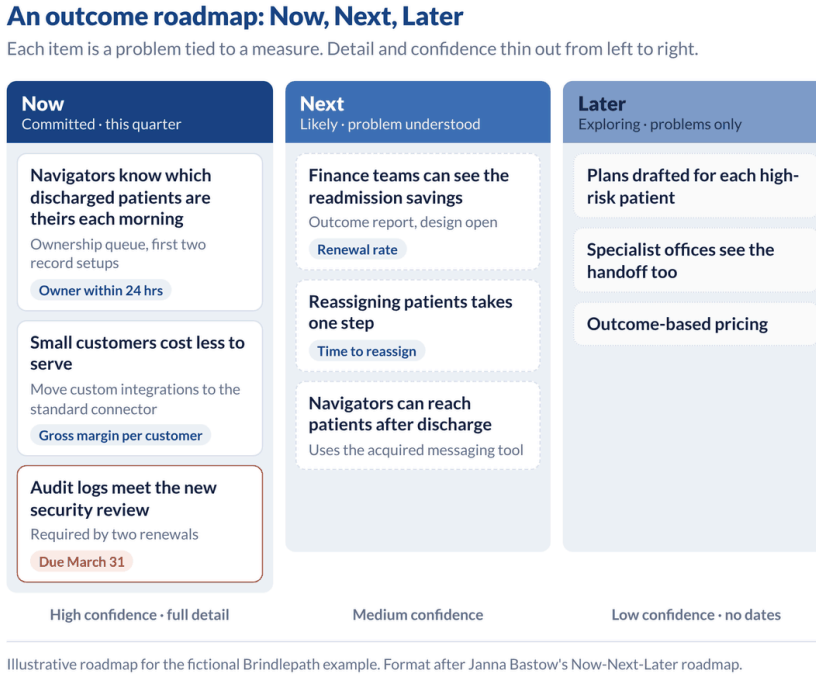
The outcome roadmap

A roadmap's job is to communicate direction and sequencing, and to support decisions. It is not a release plan, and it is not a contract.

The simplest and most useful format I know is **Now, Next, Later**, a format associated with Janna Bastow and the team at ProdPad.[2] Instead of dates on a timeline, items sit in three horizons:

- **Now:** what the team is working on, with high confidence and detail.

- **Next:** what comes after, with medium confidence; problems understood, solutions still open.
- **Later:** what the team expects to tackle, with low confidence and no detail; mostly problems, not solutions.



> Figure 5.1. Brindlepath's outcome roadmap in the Now, Next, Later format. Each item is a problem tied to a measure. Confidence drops, and detail thins, from left to right.

Three features make this format work.

Every item is a problem or an outcome, not a feature. "Navigators can confirm ownership of discharged patients within 24 hours" leaves room for the team to find the best solution. "Build a discharge dashboard" does not.

Confidence is visible. Moving from Now to Later, the detail thins out on purpose. Stakeholders can see what is committed and what is a direction.

Dates appear only where they are real. Some commitments need dates: a regulatory deadline, a contractual obligation, a conference launch. Put those dates on the roadmap explicitly. Everything else is ordered, not dated.

Different roadmaps for different audiences

One roadmap rarely serves everyone. The board wants strategy and outcomes. The sales team wants to know what they can promise and when. Engineering needs sequencing and dependencies. Customers want to know whether their problem is being addressed.

Keep one source of truth and present it differently for each audience. Never show customers anything you would be uncomfortable being held to, and never give sales a date that engineering has not agreed to. The fastest way to lose a team's trust is to promise their work to a customer without asking them.

Prioritization frameworks, and when to use them

Frameworks do not make decisions. They make the reasoning behind a decision visible, so it can be challenged and improved. Choose the framework that fits the decision.

RICE scores each option on Reach, Impact, Confidence and Effort: (Reach × Impact × Confidence) ÷ Effort. It was popularized by Intercom and works well for comparing many independent features where reach is measurable.[3] Its weakness: it rewards things that touch many users a little over things that matter deeply to a few, and in business software the "few" are often the economic buyers.

Weighted scoring rates each option against criteria you choose, such as strategic fit, customer value, revenue impact, risk reduction and effort, each with a weight. It is flexible and makes strategy explicit. Its weakness: whoever sets the weights sets the answer.

Kano analysis sorts features into basic expectations (customers are angry if they are missing, but not delighted if present), performance features (more is better) and delighters (unexpected value). It is excellent for deciding what

must be in a release and what will differentiate it.

Cost of delay asks how much value is lost for each week a feature is not available. Divided by duration, it gives *weighted shortest job first*, a sequencing rule popular in large organizations.[4] It is the right tool when timing matters: a regulatory deadline, a competitor launch, a renewal at risk.

The same backlog can come out in a very different order depending on the framework. Figure 5.2 shows five Brindlepath candidates ranked both ways, using illustrative scores.

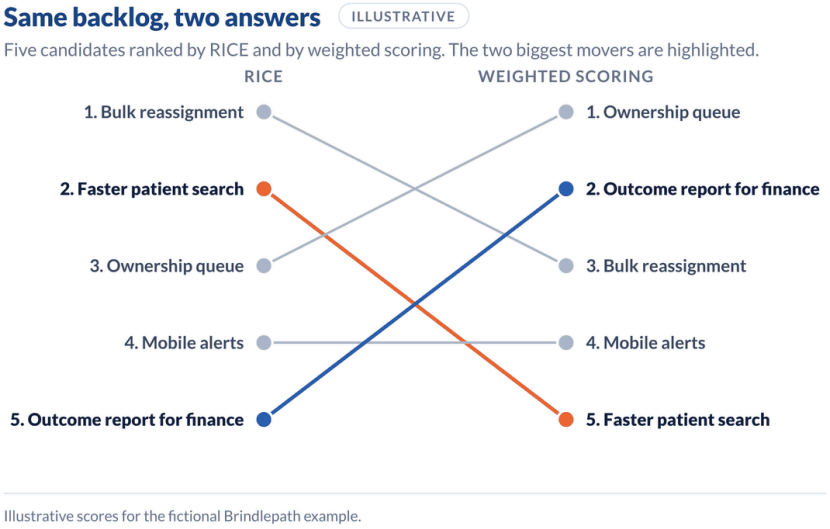


Figure 5.2. The same five candidates, ranked by RICE and by weighted scoring (illustrative). Items that touch many users rise under RICE; items that serve strategy and the buyer rise under weighted scoring.

The lesson is not that one framework is right. It is that the choice of framework is itself a strategic choice, and should be made openly. If two methods disagree sharply on an item, that item deserves a conversation, not a formula.

The opportunity solution tree

For discovery-heavy work, Teresa Torres's opportunity solution tree is a useful complement to scoring.[5] It starts with the desired outcome at the top, branches into the customer opportunities (needs, pain points, desires) that could drive it, and only then into candidate solutions and the experiments that test them. It prevents the most common prioritization error of all: comparing solutions before agreeing on the problem.

The loudest voice in the room

Every product leader faces the same pressures: the CEO's idea, the biggest customer's demand, the deal that will close "if we just add this one thing." Prioritization frameworks help, but the real skill is handling these conversations well.

For the executive's idea: treat it as a hypothesis, like any other. Ask what problem it solves and what evidence would show it is working, and put it through the same discovery as everything else. Executives usually respect a product leader who takes their idea seriously enough to test it.

For the biggest customer: separate the need from the request, as in Chapter 1. Then ask whether the need is shared by other customers in your target segment. If it is, it may belong on the roadmap. If it is unique, consider whether it is better solved by configuration, a services engagement or a partner, rather than core product.

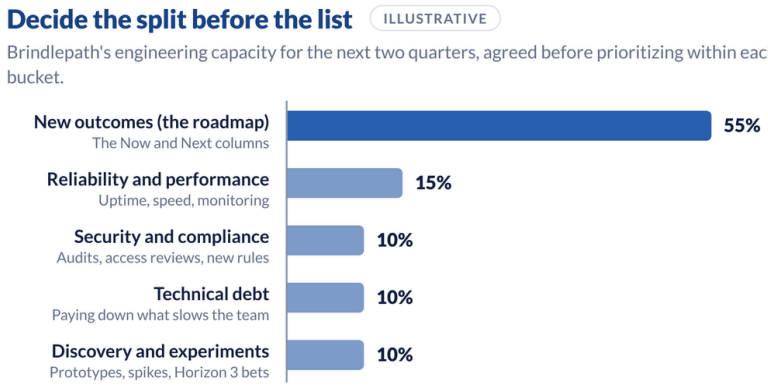
For the deal that needs one more feature: ask how many deals the feature has really decided in the past year. Sales teams are sincere, but every lost deal has many reasons, and the missing feature is the easiest one to name. Track requests by deal and outcome, and the pattern usually becomes clear.

When you say no, say it with reasons and an alternative: "We're not building that this half, because our focus is post-discharge ownership. Here's what we can do instead, and here's when we'll revisit it." A clear, reasoned no builds more trust than a vague maybe.

Making room for what keeps the lights on

A roadmap full of new features eventually collapses under its own weight. Technical debt, security work, performance, and maintenance of what already exists all need capacity, and they rarely win a head-to-head scoring contest against a shiny feature.

The fix is to allocate capacity explicitly before prioritizing within each bucket. Figure 5.3 shows the split Maya agreed with Dev Raman for Brindlepath's next two quarters.



> Figure 5.3. Capacity allocation for two quarters (illustrative). Deciding the split first stops new features from quietly consuming the time for reliability and security.

The exact split depends on the product's stage and health. A young product may put most of its capacity into new outcomes; a mature product with aging infrastructure may need far more for reliability. What matters is that the split is a decision made in daylight, reviewed each quarter, rather than the result of a hundred small trade-offs nobody saw.

Brindlepath in practice: one hard no

Six months after the new roadmap, Brindlepath's largest prospect, a health system that would have doubled the company's revenue in its region, asked for a patient-facing mobile app as a condition of purchase. The request came with

a deadline: the app had to be live within six months.

The sales team argued for it, reasonably: the deal was large, and a patient app was on the "Later" list anyway. Maya ran the request through the same process as everything else. She asked what problem the app would solve for the prospect, and learned that the real need was a way to confirm patients' contact details and preferred call times after discharge. She checked the evidence: none of Brindlepath's existing customers had asked for an app, and two had said patients rarely downloaded them. She estimated the cost: at least two-thirds of a team for six months, displacing the standard connector that every existing customer needed.

The answer was no to the app, and yes to the need. Brindlepath offered a short text-message confirmation flow, built on the recently acquired messaging tool, which could be live in eight weeks and served the same purpose. Maya joined the sales call herself to explain the reasoning: the prospect's patients would get what they needed sooner, and the prospect would be buying a product whose roadmap was not hostage to any single customer, including them.

The prospect signed, three months later, without the app. Not every hard no ends that well. But a reasoned no with an alternative ends well far more often than sales teams fear, and a roadmap that bends to every large request does not stay a roadmap for long.

Common mistakes

Treating the roadmap as a promise. Dates on features become commitments customers hold you to, whether or not the features turn out to matter. Commit to outcomes and to the few dates that are genuinely required.

Letting the backlog become a museum. A backlog of hundreds of items is not a plan. Archive anything no current customer problem supports; the important items will come back.

Scoring precisely, deciding badly. Frameworks produce numbers with decimal places that hide guesses. Use scores to structure the conversation, not to end it.

Saying maybe instead of no. “We’ll look at it” sounds kind and creates false expectations. A clear no, with reasons and an alternative, builds more trust.

Starving the foundations. Without an explicit capacity allocation, reliability, security and debt lose every head-to-head contest with new features, until an outage or breach forces the issue.

One roadmap for every audience. The board, sales and engineering need different views. A single document trying to serve them all usually serves none of them well.

Try this

Export your current roadmap and backlog. For each item, write the problem it solves and the measure it should move. Any item where you cannot do both goes into a “needs a problem” list for review. Then rebuild the roadmap in the Now, Next, Later format, with only problem-framed items, and mark the commitments that genuinely need a date. Finally, agree a capacity split with your engineering lead for next quarter, and write it at the top of the roadmap where everyone can see it.

How AI changes this

AI can do the bookkeeping. An assistant can de-duplicate a sprawling backlog, link requests to the customers and deals that raised them, draft RICE or weighted scores from available data, and summarize the evidence behind each item. Maya’s 212-item cleanup would take an afternoon today rather than a month. Treat the scores as a first draft. The model does not know your strategy unless you tell it, and it does not know which customer is about to churn.

AI creates roadmap pressure. Since 2023, almost every product team has faced demands to “add AI.” Some of those demands are right; many are a solution looking for a problem. Run AI features through the same discipline as any other: what outcome will this change, for whom, and how will we know? An AI feature that does not move a measure is an expensive demo.

AI features need different confidence. A traditional feature either works or it does not. An AI feature works some of the time, at a quality that varies with the input. Put AI items on the roadmap with explicit quality targets and evaluation plans, not just a ship date. Chapter 17 covers how.

AI can help you say no. Ask an assistant to write the case against your top three priorities, or to find the items on your roadmap that do not connect to your stated strategy. It is easier to hear an uncomfortable argument from a machine than from a colleague, and easier still to act on it before the planning meeting.

Chapter checklist

- [ ] Is every item on your roadmap framed as a problem or outcome, not a feature?
- [ ] Does your roadmap show confidence levels, with dates only where commitments are real?
- [ ] Do you have one source of truth, presented differently for each audience?
- [ ] Have you chosen a prioritization framework deliberately, and do you know its biases?
- [ ] Do you discuss items where two methods disagree, instead of trusting one formula?
- [ ] When you say no, do you give reasons and an alternative?
- [ ] Have you allocated capacity for reliability, security and debt before prioritizing features?
- [ ] Do AI items on your roadmap have quality targets and evaluation plans?
- [ ] Have you archived backlog items that no current customer problem supports?

---

1. Joshua Seiden, *Outcomes Over Output: Why Customer Behavior Is the Key Metric for Business Success* (Sense & Respond Press, 2019).

2. Janna Bastow developed the Now-Next-Later roadmap format and popularized it through ProdPad, the product-management software company she co-founded.

3. Sean McBride, “RICE: Simple prioritization for product managers,” Intercom blog, 2016.

4. Donald G. Reinertsen, *The Principles of Product Development Flow: Second Generation Lean Product Development* (Celeritas Publishing, 2009).

5. Teresa Torres, *Continuous Discovery Habits* (Product Talk, 2021).

Chapter 6. The Numbers: P&L, Unit Economics and Pricing

“Price is what you pay. Value is what you get.”

— Warren Buffett

What does a seat cost us?

In Maya’s second month, Brindlepath’s chief financial officer, Priya Nair, asked her a question in the hallway that she could not answer: “What does it cost us to serve one navigator seat for a year?”

Maya knew the price: Brindlepath charged per navigator seat, plus a platform fee per health system. She knew roughly what engineering cost. She did not know how much of the hosting bill, the support team, the implementation engineers and the integration maintenance each customer consumed. Nobody did. When Priya and Maya worked it out together over the following two weeks, they found that Brindlepath’s three smallest customers cost more to serve than they paid, mostly because each had needed a custom integration that the company now maintained for free.

That discovery changed the roadmap more than any customer interview. It also changed the way the CEO, the board and the sales team listened to Maya. A product leader who can explain the economics of a decision is a different kind of colleague from one who can only explain its features.

This chapter is the financial literacy that every product leader needs. It is not accounting. It is the handful of numbers that tell you whether your product is a good business.

Reading a SaaS P&L

A profit and loss statement, or P&L, shows revenue and costs over a period. For a software company, the structure is usually the same.

Revenue. For subscription businesses, the most watched figure is **annual recurring revenue (ARR)**: the yearly value of active subscriptions. One-time revenue, such as implementation fees, is reported separately, because it does not recur.

Cost of revenue, also called cost of goods sold (COGS). The costs of delivering the service: hosting and infrastructure, third-party software embedded in the product, customer support, implementation and integration work, and, for AI products, model and inference costs.

Gross profit and gross margin. Revenue minus cost of revenue. Gross margin, gross profit as a percentage of revenue, is the single most important number for judging whether a software product is a good business. Mature software companies typically target gross margins well above 70 percent; services-heavy or infrastructure-heavy products run lower.

Operating expenses. Sales and marketing, research and development (where most product and engineering salaries sit), and general and administrative costs.

Operating income. Gross profit minus operating expenses. Many growing software companies run at an operating loss by choice, investing in growth. That is a strategy, not a failure, as long as the unit economics underneath are sound.

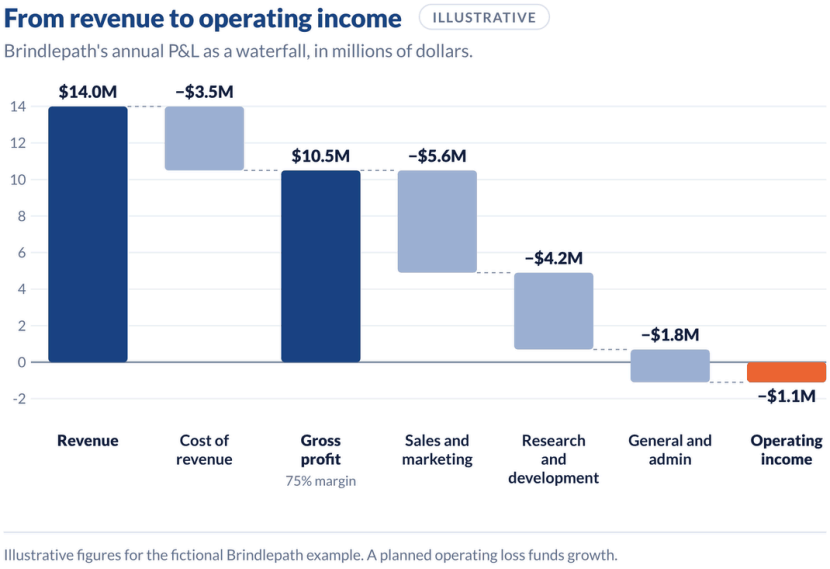


Figure 6.1. Brindlepath's annual P&L as a waterfall (illustrative). Every product decision touches one of these bars: what it adds to revenue, what it adds to the cost of delivering, and what it costs to build and sell.

Figure 6.1 shows Brindlepath's P&L in simplified form. The product leader's influence is everywhere in it. Features that reduce support tickets raise gross margin. Integrations that need custom work for each customer lower it. A product that sells itself reduces sales and marketing cost. A product that needs a six-month implementation increases cost of revenue and delays revenue recognition.

Unit economics: is each customer worth it?

The P&L tells you how the business did. Unit economics tell you whether each new customer makes the business better or worse. Five measures matter most.

Customer acquisition cost (CAC). Total sales and marketing spend in a period, divided by the number of new customers won. In enterprise healthcare, where sales cycles run nine to eighteen months and involve many stakeholders, CAC is high.

Annual gross profit per customer. Average annual revenue per customer multiplied by gross margin. This is what a customer contributes toward paying back acquisition cost and funding everything else.

CAC payback period. CAC divided by monthly gross profit per customer: how many months it takes to earn back the cost of winning the customer. Shorter is better; investors often look for well under two years.

Retention. *Gross revenue retention* is the share of last year's recurring revenue you kept from existing customers, ignoring upsells. *Net revenue retention (NRR)* adds expansion: upsells, extra seats, new modules. NRR above 100 percent means existing customers grow faster than others leave, which is the engine of the best software businesses.

Customer lifetime value (LTV). Gross profit per customer per year multiplied by the expected years a customer stays. The ratio of LTV to CAC is a common shorthand for the health of a growth model; a ratio around three or higher is a frequently quoted rule of thumb.

A worked example

Here is Brindlepath's typical mid-sized customer after Maya and Priya finished their analysis. The numbers are illustrative.

- 30 navigator seats at $1,800 per seat per year, plus a $36,000 platform fee: **$90,000 annual recurring revenue**.
- Cost to serve, including hosting, support, integration maintenance and model costs: $22,500 a year, so **gross margin is 75 percent** and gross profit is $67,500 a year, or $5,625 a month.
- Acquisition cost, including a long sales cycle and a pilot: **$120,000**.
- **CAC payback** is therefore $120,000 ÷ $5,625, or about **21 months**.
- With gross revenue retention of 90 percent, the average customer stays about ten years, giving a lifetime gross profit of roughly $675,000 and an **LTV-to-CAC ratio of about 5.6**.

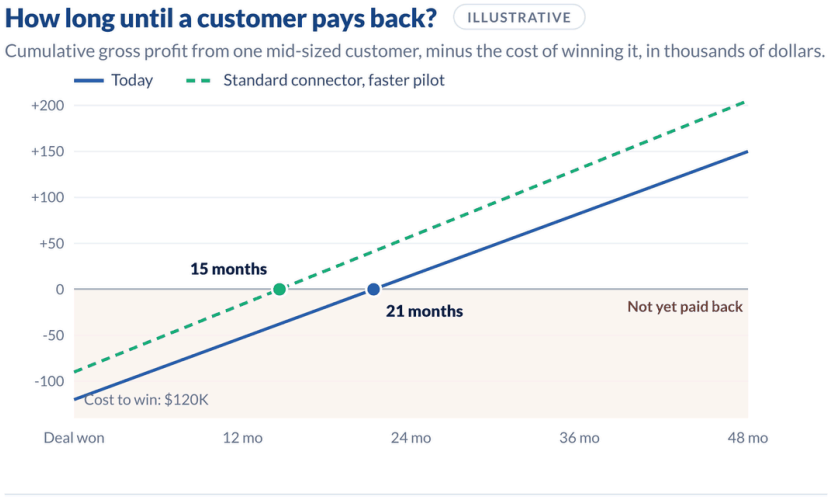


Figure 6.2. One customer's cumulative cash, from the day the deal is won (illustrative). The curve starts deep in the red and crosses zero at about 21 months. Product choices move both the depth of the dip and the slope of the climb.

Figure 6.2 shows why product decisions are financial decisions. A self-serve pilot that shortens the sales cycle makes the dip shallower. Fewer custom integrations raise gross margin and steepen the climb. A second module that customers buy in year two bends the curve upward. A clunky onboarding that causes churn in year three cuts it off before it pays back.

The three small customers Brindlepath discovered were losing money for one reason: their custom integrations had pushed cost to serve so high that gross margin was close to zero. Their curves never crossed the line. The fix was not to fire them; it was to move their integrations onto Brindlepath's standard connector, which became the first item on the Now column of the roadmap.

Pricing: the most underused lever

Pricing is one of the most powerful levers a product leader has, and one of the least studied. In a widely cited analysis of 2,463 companies, Michael Marn and Robert Rosiello found that a 1 percent improvement in price, with volume

unchanged, increased operating profit by 11.1 percent on average, far more than the same improvement in variable cost, volume or fixed cost.[1] Later McKinsey analyses of larger samples put the effect somewhat lower, at around 8 to 9 percent, but the conclusion holds: small changes in price have outsized effects on profit.

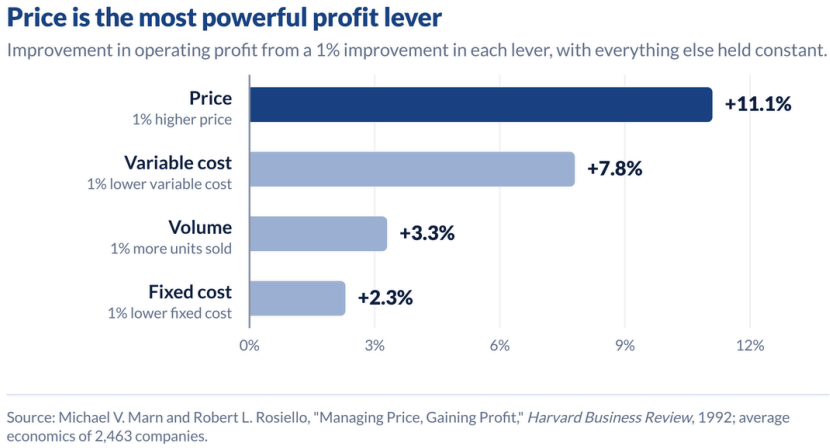


Figure 6.3. Operating profit improvement from a 1 percent improvement in each lever, average of 2,463 companies. Price is the most powerful lever by a wide margin. Source: Marn and Rosiello, Harvard Business Review, 1992.

Price to value, not to cost

There are three broad ways to set a price. **Cost-plus** pricing adds a margin to what the product costs to deliver. It is simple and almost always leaves money on the table, because customers do not care about your costs. **Competitor-based** pricing anchors to what alternatives charge. It is necessary context, but it makes you a follower. **Value-based** pricing starts with the value the customer receives and captures a fair share of it.

For Brindlepath, value was measurable. A mid-sized health system with meaningful readmission penalties, a few hundred high-risk discharges a month and a goal of preventing even a modest number of readmissions could point to savings and avoided penalties many times the price of the software. That calculation, built with the customer’s own numbers, became the heart of every

proposal. In the United States, the Medicare Hospital Readmissions Reduction Program can reduce a hospital's base Medicare payments by up to 3 percent for excess readmissions,[2] which gave finance teams a concrete number to put in the model.

The pricing metric

The **pricing metric** is the unit you charge for: per user, per seat, per transaction, per patient, per facility, per outcome. A good metric has three properties: it grows as the customer gets more value, it is easy to understand and predict, and it is hard to game.

Brindlepath's per-navigator seat price was easy to understand but had a flaw: as the product made navigators more efficient, customers needed fewer of them, so success shrank revenue. The company added a per-health-system platform fee scaled by annual discharges, which grew with the size of the problem the product solved. Finding the right metric is often worth more than finding the right price.

Packaging

Most software is sold in tiers, often three, sometimes called *good, better, best*. Packaging decides which features go in which tier. Two principles help.

- **Put the features that define your best-fit customer in the middle tier,** and price it to be the obvious choice. The top tier exists partly to make the middle look reasonable.
- **Use features to fence tiers that larger customers genuinely need,** such as advanced security, audit reporting, single sign-on and integrations, rather than artificially limiting the core experience.

Pricing deserves the same discovery as features: interviews about willingness to pay, tests of different packages with new prospects, and careful analysis of win and loss rates by price point.

Making the business case

When you propose a significant investment, a new product, a major feature or an integration, bring a business case. It does not need to be long. It needs five things:

1. **The problem and the customers who have it,** from your discovery work.
2. **The revenue effect:** new customers won, expansion, churn prevented, price increase supported. Show your assumptions.
3. **The cost to build:** engineering, design and product time, plus any third-party costs.
4. **The cost to run:** hosting, support, maintenance, model costs, partner fees. This is the part most business cases forget.
5. **The risks and how you will know:** what has to be true, and the measure you will check after launch.

Show a range, not a single number. A case that says "between $400,000 and $1.2 million of new ARR in the first year, most likely around $700,000, if pilots convert at our historical rate" is more credible, and more useful, than one that says "$1 million."

Brindlepath in practice: changing the pricing metric

As Chapter 6 described, Brindlepath's per-seat price had a flaw: the more efficient the product made navigators, the fewer seats customers needed. At one customer, a successful rollout had let the health system cover more patients with the same team, and its finance department asked, reasonably, whether it could reduce seats at renewal.

Maya and Priya spent a quarter on the problem. They interviewed finance leaders at eight customers about how they budgeted for the product and what they thought they were paying for. The answers were consistent: customers thought of Brindlepath as paying for covering their high-risk patients, not for navigator logins.

The new model kept a smaller per-seat price, to cover the cost of each user, and moved most of the value into a platform fee scaled to the number of high-risk discharges the product covered each year. For a typical customer, the total price stayed about the same in the first year. As customers expanded coverage to more hospitals and more patients, revenue grew with the value delivered, regardless of how many navigators they employed.

The team tested the model with new prospects before offering it to existing customers, and gave existing customers the choice to switch at renewal. Most did, because it made their budget easier to explain: the cost tracked the problem they were solving. Net revenue retention rose over the following year, and the conversation about reducing seats disappeared.

The lesson is not that platform fees are better than seats. It is that the pricing metric should grow with the value customers receive, and that finding it took the same discovery work as any feature.

Common mistakes

Pricing on cost. Customers do not care what your product costs to build. Cost-plus pricing systematically leaves money on the table for products that create real value.

Ignoring the cost to serve. Features that require custom work, heavy support or expensive AI calls can make customers unprofitable while revenue looks healthy.

Discounting to close. Deep discounts win this quarter's deal and set expectations for every renewal. Discount scope or terms before price, and attach discounts to commitments.

Choosing a pricing metric that punishes success. If customers pay less as your product makes them more efficient, your growth works against you.

Business cases that forget running costs. A case that counts only the cost to build will overstate the return of almost everything.

Single-number forecasts. One revenue number invites false precision. A range with explicit assumptions is more honest, and more useful for decisions.

Try this

Pick one typical customer and calculate, with help from finance if you need it, their annual revenue, their annual cost to serve (hosting, support, implementation, integration maintenance and AI costs), their gross margin, the cost of winning them and their payback period. Then repeat the exercise for your smallest customer and your most demanding one. Compare the three. Most product leaders who do this find at least one surprise, and at least one roadmap item that would change the numbers more than any new feature.

How AI changes this

AI changes your cost structure. Traditional software has very low marginal costs: serving one more user costs almost nothing. AI features add a real variable cost for every request, paid to a model provider or for your own infrastructure. A feature that summarizes every patient record every morning can quietly add meaningful cost per seat. Track AI costs per customer, per feature and per request from day one, and put them in your gross margin. Products that lean heavily on AI often run at lower gross margins than traditional software for exactly this reason, so plan for it rather than discovering it.

AI changes pricing. When software does work that a person used to do, pricing per seat can undercut the value you create, as Brindlepath found. Many AI products are experimenting with usage-based pricing (per request or per task), outcome-based pricing (per resolved case or prevented readmission) or hybrid models. Each shifts risk between you and the customer. Outcome-based pricing is compelling to buyers and hard to administer; agree in advance exactly how outcomes will be measured.

AI makes analysis cheap. An assistant can build a pricing model, a cohort analysis or a business-case spreadsheet in minutes, and run the sensitivity analysis you would never have had time for. Use it. Then check the formulas,

because a confident spreadsheet with a broken formula is worse than none.

Chapter checklist

- [ ] Do you know your product's gross margin, and what drives it?
- [ ] Do you know the cost to serve a typical customer, including support, integration and AI model costs?
- [ ] Can you state your CAC, payback period, gross and net revenue retention, and LTV-to-CAC ratio?
- [ ] Have you found any customers who cost more to serve than they pay?
- [ ] Is your price based on the value customers receive, using their own numbers?
- [ ] Does your pricing metric grow as customers get more value?
- [ ] Does every major proposal include the cost to run, not just the cost to build?
- [ ] Do your business cases show a range and the assumptions behind it?
- [ ] Are AI costs tracked per customer and per feature, and included in gross margin?

---

1. Michael V. Marn and Robert L. Rosiello, "Managing Price, Gaining Profit," *Harvard Business Review*, September–October 1992. The figures are based on the average economics of 2,463 companies in the Compustat database. Later McKinsey analyses of the Global 1200 and the S&P 1500 put the effect of a one-percentage-point price improvement at roughly 8 to 9 percent of operating profit.
2. Centers for Medicare & Medicaid Services, Hospital Readmissions Reduction Program. The maximum payment reduction is 3 percent of a hospital's base operating diagnosis-related group payments.

CHAPTER 7

Managing a Product Portfolio

“Management has no choice but to anticipate the future, to attempt to mold it, and to balance short-range and long-range goals.”

— Peter Drucker

From one product to four

A year after the offsite, Brindlepath bought a small company that made a patient-messaging tool. The logic was sound: Brindlepath’s customers kept asking for a way to reach patients after discharge, and building it from scratch would have taken a year. Overnight, Maya went from leading one product to leading four: the core care-transitions product, an outcomes-reporting module that had grown up inside it, the acquired messaging tool, and the old scheduling product, which still had paying customers despite no longer being part of the strategy.

Each product had its own customers, its own technology, its own team and its own champion on the leadership team. Each wanted more investment. In her first portfolio review, Maya realized that the hardest question was no longer *what should this product do next?* It was *how much should we invest in each product, and what should we stop?*

That question is the subject of this chapter. Even if you manage a single product today, the habits here apply to the features and modules inside it, and they will matter the day your company grows a second product.

Products have life cycles

Every product moves through stages, and each stage calls for a different kind of investment and a different measure of success.

- **Introduction.** The product is finding product-market fit. Revenue is small; learning is the main output. Measure learning velocity and early signs of retention.
- **Growth.** The product has found fit and is scaling. Invest heavily in reach, onboarding and the features that unlock larger customers. Measure growth rate, net revenue retention and acquisition efficiency.
- **Maturity.** Growth slows; the product is a reliable revenue and margin engine. Invest in efficiency, reliability, pricing and adjacent expansion. Measure margin, retention and expansion.
- **Decline.** The market or technology has moved on. Invest only to keep customers safe and satisfied while you plan the exit. Measure cost to serve and migration progress.

The most common portfolio mistake is to manage every product the same way: the same planning cycle, the same metrics, the same expectations. A product in introduction judged on revenue will be killed before it has a chance. A product in maturity judged on growth will be pushed into features nobody needs. A product in decline that keeps its full team will consume resources the growth products badly need.

Mapping the portfolio

A portfolio map makes the investment conversation concrete. There are many versions. The classic Boston Consulting Group growth-share matrix plots products by market growth and relative market share.[1] For software portfolios, I find it more useful to plot each product by its revenue growth and its gross margin, with the size of each bubble showing its revenue.

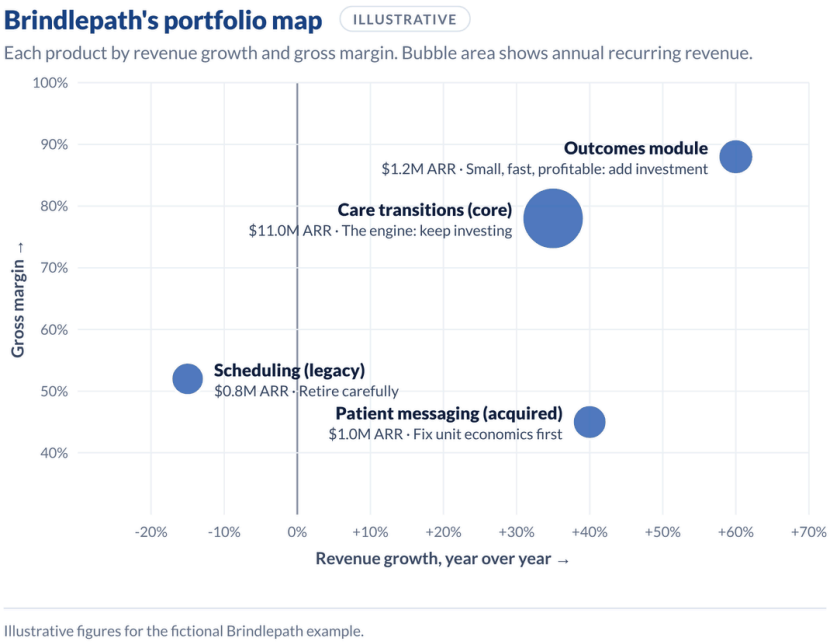


Figure 7.1. Brindlepath's four products by revenue growth and gross margin; bubble size shows annual recurring revenue (illustrative). The map turns four separate arguments into one picture.

Figure 7.1 shows Brindlepath's portfolio. The core care-transitions product is large, growing and healthy: the engine. The outcomes module is small, growing fast and highly profitable, because it reuses the core product's data: a candidate for more investment. The messaging tool, newly acquired, is growing but at a low margin, because it pays a third party for every text message sent: it needs its unit economics fixed before it scales. The scheduling product is shrinking and barely profitable: a product to retire carefully.

None of those conclusions is automatic. The map does not make the decision. It replaces four separate pitches, each delivered with passion by a product's champion, with one shared picture that the leadership team can argue about productively.

Three horizons

Portfolio thinking is also about time. The three-horizons model, introduced by Mehrdad Baghai, Stephen Coley and David White, sorts work by how far it is from today's core business.[2]

- **Horizon 1** extends and defends the core business: improvements to the products that pay the bills today.
- **Horizon 2** builds emerging businesses that are starting to generate revenue and could become the next core.
- **Horizon 3** creates options for the future: research, experiments and early bets that may not pay off for years, if ever.

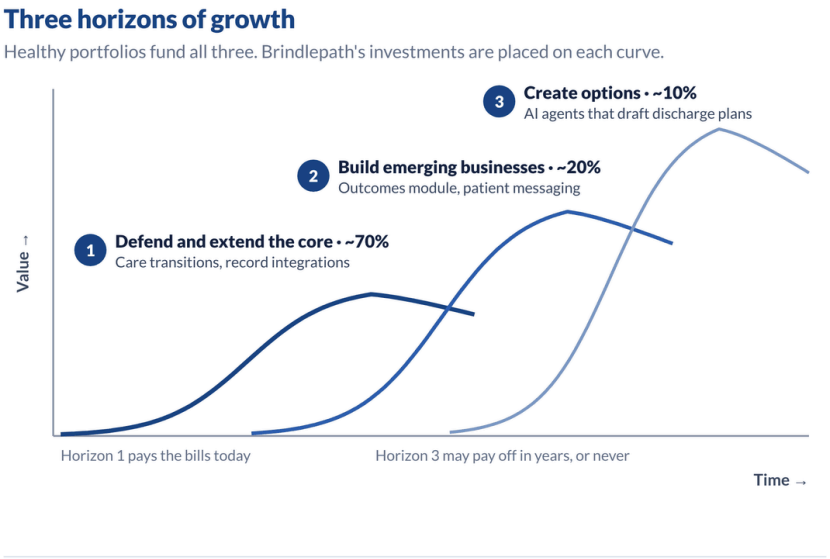


Figure 7.2. Three horizons of growth, with Brindlepath's investments placed on each (illustrative). Healthy portfolios fund all three; struggling ones spend everything on Horizon 1.

Companies under pressure tend to pour everything into Horizon 1, because it has the clearest return and the loudest customers. That works until the core business matures, at which point there is nothing ready to replace it. A deliber-

ate allocation across horizons protects the future from the present. A frequently cited guideline, associated with Google in the 2000s, puts roughly 70 percent of effort on the core, 20 percent on adjacent growth and 10 percent on transformational bets. The exact split matters less than making it explicit and reviewing it regularly.

For Brindlepath, Horizon 1 was the core care-transitions product and its record integrations. Horizon 2 was the outcomes module and the messaging tool. Horizon 3 was a small team exploring AI agents that could draft a post-discharge plan for each high-risk patient for a navigator to review, a bet that, if it worked, could change the economics of care coordination entirely.

Platform and products

As a portfolio grows, the same capabilities appear in several products: user management, security, integrations, data pipelines, notifications, reporting, AI services. Building each one separately in every product is expensive and produces inconsistent experiences.

The usual answer is a **platform**: shared services that several products build on. Brindlepath's most valuable platform capability turned out to be its record integration layer. Once it worked for the core product, the outcomes module and the messaging tool could use the same patient data without their own integrations, which cut implementation time for customers buying more than one product.

Platform work has a particular trap: it is easy to build a platform that nobody asked for, designed for imagined future products. The healthiest approach treats the platform as a product with internal customers. Extract shared capabilities from real products when two or more need them, rather than building them speculatively in advance.

Cross-selling and bundling

A portfolio creates new commercial options: selling additional products to existing customers, and bundling products together. Both can lift net revenue retention significantly, because selling to a customer who already trusts you is

far cheaper than winning a new one.

But a bundle is only as good as the integration between its parts. Customers notice quickly when “the suite” is really three products with three logins, three data models and three support teams. Before you bundle, make sure the products share identity, data and design at least well enough that using them together is easier than using them apart.

Retiring a product well

Sooner or later, every portfolio needs to retire something. Brindlepath’s scheduling product had forty paying customers, a dwindling team and no future in the strategy. Keeping it alive cost more each year in maintenance and security work; shutting it down carelessly would damage trust with customers who also bought the core product.

A responsible retirement has five steps.

1. **Decide and explain why.** Write down the reasons, the timeline and the alternatives customers will have. Share it internally first, so that sales and support can answer questions consistently.
2. **Give real notice.** For business software, twelve months is a reasonable minimum; in healthcare, where changing a workflow requires training and sometimes clinical review, longer may be needed.
3. **Offer a path.** Migration to another of your products, a partner’s product or an export of data in a usable format. Help customers move, especially your best ones.
4. **Handle data carefully.** Regulated data has retention and deletion requirements. In healthcare, records may need to be kept for years after the product is retired. Plan how data will be returned, retained or destroyed, and document it.
5. **Free the capacity deliberately.** Decide in advance where the team and budget will go. Otherwise, the savings from retirement quietly disappear into everything else.

Brindlepath migrated most scheduling customers onto a partner's scheduling tool connected to the core product, kept a small number on an extended support contract, and moved the scheduling team into the record integration platform, where their knowledge of the health systems' systems was invaluable.

Running portfolio reviews

A portfolio needs a regular rhythm of decisions, separate from the planning of individual products. Many companies hold a portfolio review each quarter. A good review covers:

- **Each product's stage and its measures** of success for that stage, not one common scorecard.
- **The portfolio map,** updated with the latest numbers.
- **The horizon allocation,** planned versus actual.
- **Shared platform needs** and conflicts between products.
- **Decisions:** where to add investment, where to reduce it, what to stop, and what to explore.

The output is a short list of decisions with owners, not a set of status updates. If a portfolio review ends without any product getting more or less than it had before, ask whether anything was actually reviewed.

Brindlepath in practice: the first portfolio review

Maya's first formal portfolio review, a quarter after the messaging acquisition, ran for two hours with the CEO, CFO, head of sales and the leaders of each product. She sent the portfolio map in Figure 7.1 two days ahead, with a short page for each product covering its stage, its stage-appropriate measures and a proposed decision.

The core care-transitions product was in growth. Its measures were net revenue retention, time to first value and the North Star. Proposed decision: keep investing, with the standard connector as the priority.

The outcomes module was in introduction moving into growth: small, fast-growing and very profitable. Proposed decision: add two engineers and make it part of the standard package for new customers.

The messaging tool was in growth but with poor unit economics, because the per-message fees it paid a third party were not passed on to customers. Proposed decision: renegotiate the provider contract, add usage-based pricing for high volumes, and hold investment until gross margin passed 65 percent.

The scheduling product was in decline. Proposed decision: announce retirement with twelve months' notice, migrate customers to a partner, and move the team to the integration platform.

The discussion was lively, and not every proposal survived intact. The CEO pushed back on retiring scheduling so soon; the compromise was fifteen months' notice. The head of sales wanted the outcomes module offered at a discount; the decision was to include it in the standard package instead. But the meeting ended with a written list of decisions and owners, and the next quarter's review began by checking them. That habit, more than any framework, is what turned a collection of products into a portfolio.

Common mistakes

One scorecard for every product. Judging a new product on revenue, or a mature one on growth, pushes both in the wrong direction. Measure each product on what matters at its stage.

Funding by volume, not evidence. The product with the most passionate champion often gets the most investment. A shared portfolio map makes the conversation about evidence instead.

Starving the future. Spending everything on Horizon 1 feels responsible until the core matures and nothing is ready to replace it.

Building the platform first. Shared capabilities built speculatively, for products that do not exist yet, are rarely what those products need. Extract them from real use.

Bundling before integrating. A suite that is three logins and three data models disappoints the customers you most wanted to impress.

Never retiring anything. Products in decline quietly consume maintenance, security and support capacity for years. Retiring them well frees investment for growth.

Try this

Draw your own portfolio map. If you have one product, use its major modules or customer segments. Plot each by revenue growth and gross margin, size it by revenue, and label its life-cycle stage. Then write one sentence for each: invest more, hold, fix or retire. Bring the map to your next leadership meeting and ask whether everyone agrees with the sentences. Where they do not, you have found the conversation the portfolio review should have.

How AI changes this

AI becomes a platform capability. Many companies are building shared AI services, such as model access, retrieval over their own data, evaluation tools and safety controls, that every product can use. Treating AI as a platform avoids each team building its own and makes cost, quality and compliance manageable. It is also a natural Horizon 2 or 3 investment for most portfolios.

AI can blur product boundaries. An AI assistant that can act across several products, answering a question by drawing on scheduling, messaging and outcomes data together, can make a collection of products feel like one. That raises the value of integrated data and shared identity, and the cost of a portfolio that is really separate products under one logo.

AI can disrupt your own core. The most uncomfortable portfolio question of this era is whether an AI-native approach could replace your Horizon 1 product. Brindlepath's Horizon 3 agent experiment could, if it worked, make some of its own core workflow unnecessary. Better to discover that yourself, in a small team you control, than to read about it in a competitor's launch announcement.

AI makes portfolio analysis faster. Assistants can assemble the portfolio map, reconcile metrics across products, and summarize customer feedback product by product before each review. As always, the judgment about where to invest stays with the leadership team.

Chapter checklist

- [ ] Do you know which life-cycle stage each of your products or major modules is in?
- [ ] Is each product measured on the metrics that matter for its stage?
- [ ] Do you have a portfolio map that the leadership team discusses together?
- [ ] Is your investment split across the three horizons explicit, and reviewed regularly?
- [ ] Are shared capabilities built as a platform with internal customers, extracted from real needs?
- [ ] Before bundling, do your products share identity, data and design well enough to be easier together?
- [ ] Do you have a responsible plan for retiring products, including notice, migration and data handling?
- [ ] Do your portfolio reviews end with decisions about investment, not just status updates?
- [ ] Have you asked whether an AI-native approach could replace your own core product?

---

1. The growth-share matrix was developed by Bruce Henderson and the Boston Consulting Group in 1970.
2. Mehrdad Baghai, Stephen Coley and David White, *The Alchemy of Growth: Practical Insights for Building the Enduring Enterprise* (Perseus Books, 1999).

PART 3

Building

Chapter 8. Building and Leading the Product Team

“Great things in business are never done by one person. They’re done by a team of people.”

— Steve Jobs

The team that shipped and the team that didn’t

When Maya Smith arrived at Brindlepath, the company had two product teams of about the same size and seniority. One shipped steadily, recovered quickly from mistakes, and had a reputation with customers for listening. The other shipped in bursts, argued constantly, and had lost two engineers in six months.

Maya’s first instinct was to look at skills. The struggling team had good people: a strong product manager, an experienced designer, talented engineers. What it lacked was harder to see. In meetings, the engineers rarely questioned the product manager’s requirements, then quietly built something different. The designer learned about features after they had been specified. Nobody admitted a mistake until it reached a customer. Decisions were relitigated in the hallway after they had been made in the room.

The healthy team was not more talented. It was safer, clearer and closer. Its product manager, designer and engineering lead met with customers together, argued openly about what to build, and wrote down what they decided. When something broke, the first question was “what happened?” not “who did this?”

Maya spent more of her first year on the struggling team's dynamics than on any feature. It was the best investment she made.

The roles on a product team

A modern product team is small, cross-functional and long-lived. It owns a meaningful part of the product, a problem area or a customer journey, and has most of the skills it needs to discover, build and run solutions without waiting on other teams.

The core is the **product trio**: a product manager, a product designer and an engineering lead.[1] Together they own discovery. They talk to customers together, decide together what to test, and share responsibility for the outcome. Around the trio sit the rest of the engineers and, depending on the product, specialists in data, quality, content, security or clinical practice.

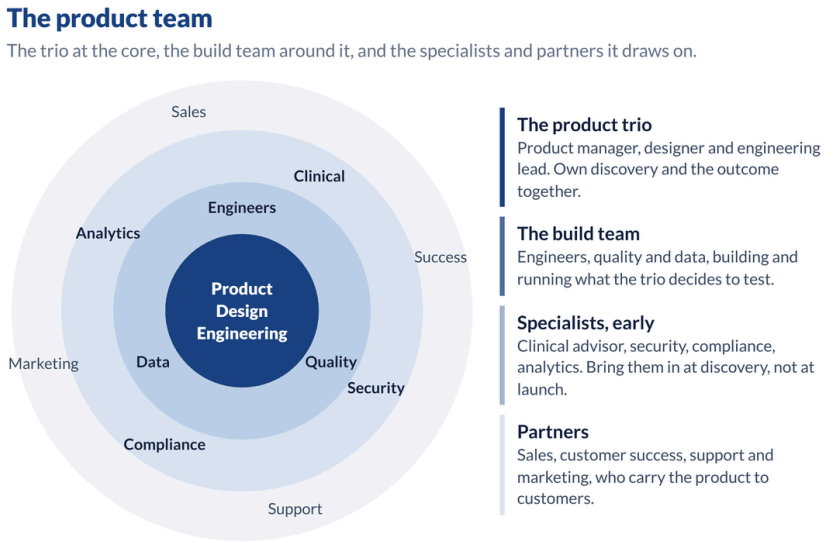


Figure 8.1. A product team: the trio at the core, the build team around it, and the specialists and partners it draws on. In regulated products, compliance and clinical experts belong in the inner rings early, not at the end.

Each role has a distinct center of gravity:

The product manager is accountable for value and viability: that the team works on the right problems, that solutions create value for customers and the business, and that the work fits the strategy. The product manager is not the boss of the team, and does not own design or engineering decisions.

The product designer is accountable for usability and the experience: that customers can understand and use what the team builds, and that it feels coherent with the rest of the product. Great designers are involved from the first customer conversation, not handed requirements to decorate.

The engineering lead is accountable for feasibility and quality: that solutions can be built, run, secured and maintained, and that the team's technical choices serve the long term. The best engineering leads shape what gets built by proposing simpler or more powerful approaches the product manager had not imagined.

Specialists bring depth the trio lacks. In healthcare, a clinical advisor, often a nurse or physician, can tell you in five minutes what would take five weeks of research to learn. A compliance partner involved early turns regulation from a late-stage veto into a design input.

The lines between roles are not walls. The best teams overlap generously: engineers who talk to customers, designers who understand the data model, product managers who can read a pull request. What matters is that accountability is clear, even when contributions are shared.

What makes a team effective

Between 2012 and 2015, Google studied over 180 of its own teams to find out why some were more effective than others. The researchers expected to find that the best teams had the best individuals, the right mix of skills or personalities, or the most experienced leaders. Instead, they found that **how** a team worked together mattered far more than **who** was on it.[2] They identified five dynamics:

1. **Psychological safety:** team members feel safe to take risks and be vulnerable in front of each other.

2. **Dependability:** team members get things done on time and to a high standard.
3. **Structure and clarity:** team members have clear roles, plans and goals.
4. **Meaning:** the work is personally important to team members.
5. **Impact:** team members believe their work matters.



> Figure 8.2. The five dynamics of effective teams. Psychological safety is the foundation the other four rest on. Source: Google re:Work, Project Aristotle.

Psychological safety stood out as by far the most important, underpinning the other four. The idea comes from Amy Edmondson's research on hospital teams, which found that the best-performing units reported *more* errors, not fewer, because people felt safe to speak up about them.[3] For a product team, psychological safety shows up as engineers who challenge a requirement before building it, designers who say "I don't think this works," and product managers who admit their hypothesis was wrong.

How a product leader builds safety

Psychological safety is not niceness. It is the confidence that you will not be punished or humiliated for speaking up with ideas, questions, concerns or mistakes. A product leader builds it through small, repeated behaviors:

- **Go first on mistakes.** Talk openly about your own wrong calls and what you learned. The team takes its cue from you.
- **Ask, then listen.** “What am I missing?” and “What would make this fail?” are the most valuable questions in product management.
- **Separate the decision from the person.** Critique ideas, not people. When something goes wrong, ask what happened in the system, not who is to blame.
- **Thank dissent.** When someone disagrees in a meeting, thank them, even when you still disagree.
- **Close the loop.** When someone raises a concern, tell them what you did about it, even if the answer is “nothing, and here’s why.”

Brindlepath’s struggling team improved within two quarters once its trio began meeting customers together and running blameless reviews after incidents. The engineers who had quietly built something different started saying so in the room.

Structure and clarity

Safety without clarity produces pleasant teams that do not ship. The other dynamics, especially structure and clarity, need deliberate work too.

Give each team a mission, not a feature list. “Make sure every high-risk discharged patient has a confirmed owner within 24 hours” gives a team room to think. “Build the ownership queue” does not.

Organize teams around customer problems or journeys, not technology layers. A team that owns “the navigator’s morning” can move faster than three teams that own the front end, the back end and the database, and must coordinate every change.

Make decision rights explicit. Who decides what to build, how it looks, how it is built, and when it ships? Write it down. A simple model is that the product manager decides *what* and *why*, design decides *how it works for people*, engi-

neering decides *how it is built*, and the trio decides *when it is good enough* together.

Keep teams stable. Teams that stay together long enough to build trust and deep domain knowledge outperform teams assembled for each project. Move work to teams, not people to work.

Hiring and growing product managers

Hiring is the most important decision a product leader makes. A great product manager can make a mediocre team good; a poor one can make a great team miserable.

When I hire product managers, I look for five things above experience or industry background:

1. **Curiosity about customers.** Do they ask about the people who use the product before they ask about the product?
2. **Clear thinking and writing.** Can they explain a complex trade-off in plain language, on paper?
3. **Judgment under uncertainty.** Given incomplete information, can they make a reasonable call and say what would change their mind?
4. **Influence without authority.** Have they moved people who did not report to them?
5. **Ownership.** Do they talk about outcomes they were accountable for, including the ones that failed?

The best interview exercises mirror the real work: a short written analysis of a product problem, a role-play of a difficult stakeholder conversation, a review of a real customer interview. Brain teasers and generic “design a product for X” questions tell you little.

Growing the team

Product management has few formal training paths, so most growth happens on the job. A product leader’s job includes building a development system:

- **Clear expectations by level,** so people know what "good" looks like as an associate, a product manager, a senior product manager and a leader.
- **Regular coaching,** focused on real decisions: reviewing a strategy document, debriefing a customer interview, preparing for a difficult conversation.
- **Stretch opportunities** with support: owning a new area, leading a launch, presenting to the board.
- **A learning habit:** shared reading, internal talks, and time for people to learn the domain deeply. In healthcare, that might mean shadowing clinicians every quarter.

Remote and hybrid teams

Most product teams now work partly or fully remotely. Google's study found that whether teammates sat together had little to do with effectiveness. What matters is that the dynamics are deliberately supported.

Remote teams need more written communication, more explicit decisions, and more intentional connection. Write decisions down where everyone can find them. Use meetings for discussion and decisions, not status updates. Overlap working hours enough for real-time collaboration. Make customer conversations visible to everyone by recording them, with permission, and sharing highlights. And invest in occasional in-person time, especially for planning and for new team members.

Brindlepath in practice: turning a team around

The struggling team from the opening of Chapter 8 did not change because of a reorganization or a new hire. It changed through a handful of practices, introduced over two quarters.

First, the trio started meeting customers together. Every week, the product manager, the designer and the engineering lead joined at least one customer conversation, and spent fifteen minutes afterward agreeing what they had

heard. Within a month, the engineers stopped quietly building something different from the requirements, because they had heard the same customers say the same things.

Second, the team adopted blameless reviews. After a production incident in which a configuration change hid patients from the queue for a morning, the review asked what in the system had allowed it, not who had made the change. The answer, a missing check in the release process, was fixed in a week. The engineer who had made the change led the review, which did more for psychological safety than any speech could have.

Third, the team wrote down its decision rights. A one-page agreement set out who decided what, and how disagreements would be escalated. Hallway relitigation dropped sharply once there was a clear place to raise a disagreement.

Fourth, Maya changed how she ran reviews. She stopped presenting conclusions and started asking questions: what are we missing, what would make this fail, what did we get wrong last time? She went first in admitting her own mistakes.

Six months later, the team's anonymous survey scores on the five dynamics had risen on every one, most on psychological safety. Its delivery measures had improved too. The two engineers who had been considering leaving stayed.

Common mistakes

Hiring for résumé, not for the work. Impressive company names say little about whether someone can write clearly, make decisions under uncertainty and influence people they do not manage. Test the real work.

Handing requirements over the wall. A product manager who writes specifications and hands them to design and engineering loses the ideas those disciplines would have contributed, and their commitment.

Mistaking harmony for safety. Teams that never disagree are rarely safe; they are usually silent. Psychological safety shows up as open disagreement about ideas.

Reorganizing too often. Every reorganization resets trust and domain knowledge. Change team structures deliberately and rarely.

Bringing specialists in at the end. Clinical, security and compliance experts consulted only before launch can only say no. Consulted at discovery, they shape a better product.

Leaving growth to chance. Product managers who never get feedback or coaching plateau early. Development needs a system, not good intentions.

Try this

At your next team retrospective, ask everyone to answer five questions anonymously on a scale of one to five, one for each dynamic in Figure 8.2: Do I feel safe taking risks on this team? Can I count on my teammates? Are our goals and roles clear? Does this work matter to me? Do I believe it makes a difference? Share the averages with the team, without names, and discuss the lowest-scoring dynamic. Agree on one change to try for a month, then ask the same five questions again.

How AI changes this

AI joins the team as a tool, and sometimes as a teammate. Product teams increasingly work alongside AI assistants and agents that draft, analyze, test and monitor. The roles in Figure 8.1 do not disappear, but each role's work changes. Engineers review more code than they write. Designers explore more options faster. Product managers spend less time on synthesis and status, and more on judgment and alignment.

Smaller teams can do more. A trio with strong AI tooling can prototype, test and ship at a pace that once required a larger team. That is an opportunity to keep teams small and focused, rather than a reason to cut them to the bone.

The constraint that remains is human: the number of customer relationships, decisions and reviews a team can handle well.

New skills matter. Every role now benefits from knowing how to instruct, evaluate and correct AI tools, and how to tell good output from plausible output. Make this an explicit part of your development system, not something people pick up by accident.

Safety matters even more. When AI writes a first draft, people need to feel safe saying "this is wrong," including when the draft came from a senior colleague's prompt. And they need to feel safe admitting when they used AI and it failed. Teams that hide their AI use learn nothing from it.

Agents need owners. If your team deploys AI agents that act in production, someone must own each one: its instructions, its permissions, its quality and its failures. Chapter 17 covers how to lead teams that include agents.

Chapter checklist

- [ ] Does each product team have a clear mission framed as an outcome, not a feature list?
- [ ] Does each team's trio of product, design and engineering do discovery together?
- [ ] Are decision rights between product, design and engineering written down?
- [ ] Are specialists, such as clinical, compliance and data, involved early enough to shape decisions?
- [ ] Do you go first in admitting mistakes, and do you run blameless reviews?
- [ ] Are teams organized around customer problems or journeys, and kept stable?
- [ ] Do your interviews test the real work: writing, judgment, influence and customer curiosity?
- [ ] Do you have clear expectations by level and regular coaching for your product managers?
- [ ] Does every AI agent your team runs in production have a named owner?

1. The term *product trio* is used by Teresa Torres in *Continuous Discovery Habits* (Product Talk, 2021).
2. Google re:Work, “Guide: Understand team effectiveness,” describing Project Aristotle, Google’s study of more than 180 teams.
3. Amy C. Edmondson, “Psychological Safety and Learning Behavior in Work Teams,” *Administrative Science Quarterly* 44, no. 2 (1999); and *The Fearless Organization* (Wiley, 2018).

Chapter 9. From Requirements to Release, with Engineering and Design

“Simplicity, the art of maximizing the amount of work not done, is essential.”

— Manifesto for Agile Software Development

The forty-page specification

Brindlepath’s first attempt at the ownership queue began the old way. A product manager on the struggling team, eager to get it right, wrote a forty-page specification. It covered every screen, every field, every edge case she could imagine. It took six weeks to write and two more to review. By the time engineering started, the design had changed twice, the pilot customer had changed its discharge workflow, and two of the edge cases had turned out not to exist.

The second attempt began with a two-page brief: the problem, the outcome measure, what was known and unknown, what was out of scope, and the open questions. The trio sketched three approaches together on the first afternoon, tested the most promising with four navigators that week, and had a working slice in a pilot customer’s hands within a month. The brief grew as the team learned, and by launch it was a clear record of what had been built and why.

The difference was not that the second team skipped the thinking. It was that they did the thinking together, in small steps, with customers in the loop, instead of alone, up front, in a document.

Requirements are a conversation, not a contract

A requirements document exists to create shared understanding across the team: what problem we are solving, for whom, how we will know it worked, and what constraints we must respect. It is not a contract between product and engineering, and it is not a substitute for talking.

The best requirements documents I have seen share five traits:

1. **They start with the problem and the outcome,** not the solution.
2. **They separate what is known from what is assumed,** and say how the assumptions will be tested.
3. **They state what is out of scope,** as explicitly as what is in scope.
4. **They are short at the start and grow with learning,** rather than being complete before work begins.
5. **They are written for the whole team,** in plain language, not for a review committee.



> Figure 9.1. A one-page product brief. It begins with the problem and the measure, names what is out of scope, and lists the open questions the team will answer as it builds.

Figure 9.1 shows the structure Brindlepath now uses for every significant piece of work. The full template is in the Appendix. The brief is the start of the conversation. User stories, designs, technical designs and test plans follow from it, written by the people closest to each.

User stories and acceptance criteria

For day-to-day work, many teams express requirements as **user stories**: short descriptions of something a user needs, written from their point of view. The classic form is: *As a [type of user], I want [some goal] so that [some reason].*

“As an outpatient navigator, I want to see which newly discharged patients are assigned to me so that I can contact the highest-risk ones first.”

A story is a placeholder for a conversation, not a complete specification. It becomes buildable through **acceptance criteria**: the specific conditions that must be true for the story to be done. Good acceptance criteria are concrete and testable:

- Patients discharged in the past 24 hours appear in the queue within two hours of discharge.
- Each patient shows risk level, discharging unit and inpatient contact.
- The queue is sorted by risk level, then by discharge time.
- Only patients assigned to the signed-in navigator, or to their team, are visible.
- Every view of a patient record is logged for audit.

That last criterion is easy to forget and impossible to add cheaply later. In regulated products, privacy and audit requirements belong in acceptance criteria from the first story, not in a separate compliance phase.

Working with design

Design is not decoration applied at the end. It is the discipline of making the product understandable, usable and coherent, and it starts at the first customer conversation.

Involve design in discovery. Designers who hear customers firsthand make better decisions than designers who receive a summary. Brindlepath's head of design, Lena Park, joined the navigator interviews in Chapter 3; her sketches from those sessions became the first prototype.

Explore before converging. Ask for several directions before choosing one. The first idea is rarely the best, and comparing options is the fastest way to find out what really matters to users.

Test with real users, early and cheaply. A paper sketch tested with three navigators teaches more than a polished mock-up reviewed by the leadership team.

Respect the design system. A shared library of components and patterns makes the product consistent, accessible and faster to build. Deviations should be deliberate, not accidental.

Design for the real environment. Brindlepath's navigators worked on aging hospital computers, often with a headset on and a phone ringing. Small fonts, low-contrast text and multi-step workflows failed in that setting even when they tested well in a quiet conference room. Accessibility, which Chapter 11 covers, is part of this: design for the full range of people who will use the product.

Working with engineering

The relationship between product and engineering defines whether a team ships well. In healthy teams, product brings the *why* and the *what*, engineering brings the *how*, and both argue productively about the trade-offs.

Bring engineers in early. The best solutions often come from engineers who know what the technology can do cheaply that the product manager assumed was hard, or vice versa. Dev Raman's suggestion to build the first version of the ownership queue on the hospital's existing discharge feed, rather than a new integration, cut two months from the timeline.

Talk in outcomes and constraints, not tasks. "We need navigators to see new patients within two hours of discharge, and it must work with these two record systems" invites better solutions than "build a polling job that runs every five minutes."

Make trade-offs visible. Every project trades scope, time, quality and cost. When something has to give, decide together what it will be, and write it down.

Protect quality. Pressure to ship always pushes against testing, security and reliability. A product leader who trades quality for speed without saying so is borrowing against the team's future. Chapter 5's capacity allocation exists so this trade-off is made deliberately.

Respect estimates as forecasts, not promises. Estimates are useful for sequencing and trade-offs. Treating them as commitments teaches teams to pad them.

From idea to release: the delivery flow

Most product teams today use some form of agile delivery: small increments of work, frequent releases, and regular reflection. The ceremonies vary, whether two-week sprints, continuous flow or something in between. The underlying flow is similar.

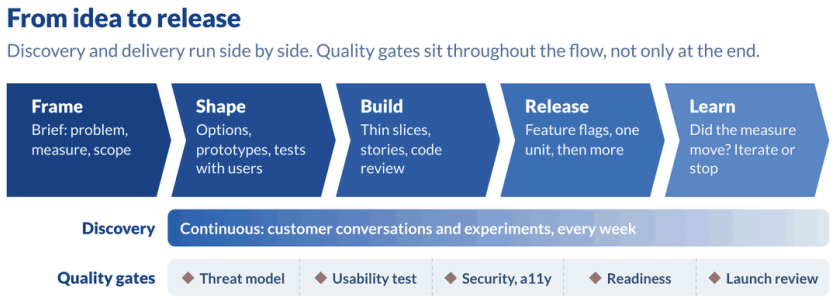


Figure 9.2. From idea to release. Discovery and delivery run side by side, and quality gates sit throughout the flow, not only at the end.

A few practices make the biggest difference.

Slice thin. Deliver the smallest piece of value that can be tested with real users, then grow it. The first slice of Brindlepath's ownership queue showed only patient name, risk level and discharge time for one hospital unit. It was enough to learn whether navigators would use it every morning.

Release often. Small, frequent releases are less risky than large, infrequent ones, because each change is easier to understand, test and undo. Research by the DevOps Research and Assessment program, summarized in *Accelerate*, found that teams with high delivery performance deploy more often *and* have lower change failure rates: speed and stability go together rather than against each other.[1]

Separate deployment from release. Feature flags let teams deploy code without exposing it, then turn it on for one customer, one unit or a small percentage of users. In healthcare, where a change to a clinical workflow may require training or approval at the customer, this is invaluable.

Define "done" to include quality. A story is not done when the code is written. It is done when it is tested, secure, accessible, documented and measurable in production.

Measure delivery, but not just speed. The four delivery measures from *Accelerate*, deployment frequency, lead time for changes, change failure rate and time to restore service, balance speed with stability. Use them to improve the system, never to compare or rank individuals.

Launch readiness

A release is not a launch. A release puts code in production; a launch puts value in customers' hands, with everything they need to use it. For significant changes, especially in regulated settings, a launch readiness review a week or two before launch saves a great deal of pain. It covers:

- **Product:** acceptance criteria met, measures instrumented, known issues documented.
- **Quality and security:** testing complete, security review done, privacy impact assessed.
- **Customers:** release notes, training materials, and, for clinical workflows, any customer approvals required.
- **Support:** support team trained, help articles ready, escalation paths clear.
- **Sales and marketing:** positioning and messages agreed, as covered in Chapter 13.
- **Rollback:** a clear plan to turn it off if something goes wrong.

The full checklist is in the Appendix.

After the release: learning

The work is not finished at launch. For every significant change, the team should know in advance which measure it expects to move, and look at it after launch. Did navigators confirm ownership faster? Did missed follow-ups fall? If not, why not?

Two practices help. A **launch review** a few weeks after release compares the result with the expectation and decides what to do next: iterate, expand, or stop. A **blameless post-incident review** after anything goes wrong asks what in the system allowed the failure, not who caused it. Both feed the team's learning and, over time, its judgment.

Brindlepath in practice: thin slices

The ownership queue was delivered in five slices over three months, each one in a pilot customer's hands within two weeks of the last.

Slice 1. For one hospital unit: patient name, risk level and discharge time, with a button to confirm ownership. No reassignment, no filtering, no reporting. The question it answered: will navigators open this every morning?

Slice 2. All units at the pilot hospital, with the discharge summary and inpatient contact attached. The question: does having the context in one place change how quickly navigators call?

Slice 3. Reassignment between navigators and teams. The question: how often does ownership change, and why?

Slice 4. Audit logging visible to the customer, role-based access by team, and the accessibility fixes found in testing. These were in the acceptance criteria from the start but were completed and verified here, before the second customer went live.

Slice 5. Outcome measures for the customer's managers: owner within 24 hours, by team and by day.

Each slice was released behind a feature flag and turned on for the pilot customer when its navigators were ready. Two slices changed significantly because of what the previous one taught. After slice 2, the team learned that navigators wanted to see whether the patient had a follow-up appointment already booked, which moved up the roadmap. After slice 3, they learned that most reassignments happened on Monday mornings for weekend discharges, which led directly to the bulk reassignment feature.

The forty-page specification from the first attempt would have built none of those changes, and several things nobody needed.

Common mistakes

Writing the whole specification up front. A complete document before any learning guarantees that parts of it will be wrong, and that the team will feel bound to build them anyway.

Leaving non-functional requirements out. Privacy, audit, accessibility, performance and security are the requirements most often forgotten and most expensive to add later.

Treating estimates as promises. Teams that are punished for missed estimates learn to pad them, and the organization loses the information it needed.

Releasing big batches. Large, infrequent releases are harder to test, harder to understand when something breaks, and harder to roll back.

Calling it done when the code is written. Work is done when it is tested, secure, accessible, documented, measurable and in customers' hands.

Never checking the result. A launch without a review of the measure it was meant to move teaches the team nothing about whether its judgment is improving.

Try this

Take the next significant piece of work your team will start and write a one-page brief using the template in Figure 9.1 and the Appendix, with the whole trio in the room, in under an hour. Pay special attention to two sections: “out of scope” and “must include.” Then write the acceptance criteria for the first story, and check that privacy, audit and accessibility appear in them. If writing the brief together surfaces a disagreement, you have saved weeks of rework by finding it now.

How AI changes this

AI drafts the paperwork. An assistant can turn a product brief into draft user stories and acceptance criteria, generate test cases from them, write release notes from merged changes, and keep the brief up to date as decisions are made. Review everything. AI-generated acceptance criteria often miss exactly the non-functional requirements, privacy, audit, accessibility and performance, that matter most in regulated products.

AI changes how code is written. AI coding assistants now write a large share of first-draft code in many teams. That makes prototyping and slicing faster, and puts more weight on code review, testing and architecture. A product leader should expect engineering to spend more time reviewing and less time typing, and should protect time for that review.

Prototypes get real. With AI design and coding tools, a working prototype connected to sample data can be built in a day. That blurs the line between discovery and delivery. Be explicit about which prototypes are throwaway experiments and which are the start of production code, because the quality bar is very different.

Requirements for AI features are different. An AI feature’s acceptance criteria are statistical: “summaries contain no fabricated facts in at least 99 percent of an evaluation set of 500 real discharge summaries, reviewed by a clinician.” They need evaluation sets, quality thresholds and monitoring in production. Chapter 17 covers this in depth.

Keep humans on the gates. AI can help at every stage of the delivery flow, but the quality gates, security review, clinical safety review and launch readiness, need accountable people.

Chapter checklist

- [ ] Does every significant piece of work start with a short brief: problem, outcome measure, scope, out of scope and open questions?
- [ ] Do user stories have concrete, testable acceptance criteria, including privacy, audit and accessibility?
- [ ] Is design involved from the first customer conversation?
- [ ] Are engineers involved early enough to shape solutions, not just estimate them?
- [ ] Do you slice work thin and release often, with feature flags to separate deployment from release?
- [ ] Does your definition of done include testing, security, accessibility and measurement?
- [ ] Do you track delivery speed and stability together, and never use them to rank individuals?
- [ ] Do significant launches have a readiness review and a rollback plan?
- [ ] Do you review each launch against the measure you expected to move?
- [ ] Are AI-generated requirements and code reviewed by people accountable for them?

---

1. Nicole Forsgren, Jez Humble and Gene Kim, *Accelerate: The Science of Lean Software and DevOps* (IT Revolution, 2018).

Chapter 10. Data, Analytics and Evidence-Based Decisions

“Not everything that can be counted counts, and not everything that counts can be counted.”

— William Bruce Cameron

Ninety-four charts

Brindlepath’s main analytics dashboard had ninety-four charts. Logins by day. Page views by screen. Average session length. Tickets by category. Each had been added by someone, for a reason, at some point. Nobody looked at most of them. When the CEO asked, at a board meeting, whether the new ownership queue was working, it took the data team three days to answer, because none of the ninety-four charts measured whether patients actually had an owner.

The problem was not a lack of data. It was a lack of agreement about which numbers mattered, and why. Maya replaced the dashboard with a single page: one measure that captured the value Brindlepath created for customers, four measures that drove it, and a handful of health checks. The page had eleven charts. People looked at it every day.

This chapter is about choosing the numbers that matter, collecting them well, and using them to make better decisions, without pretending that data can make decisions for you.

Data-informed, not data-driven

"Data-driven" is a popular phrase, and a slightly dangerous one. Data cannot tell you what you should want, which customers to serve, or whether a trade-off is worth making. It can tell you, with varying confidence, what is happening and sometimes why. The decisions are still yours.

A better phrase is **data-informed**: decisions are made by people, using evidence alongside judgment, strategy and customer understanding. Data-informed teams do three things well:

1. **They agree in advance what they will measure and why,** so they are not tempted to find a metric that flatters the result afterward.
2. **They combine quantitative and qualitative evidence.** Numbers say *what* is happening; conversations say *why*.
3. **They know the limits of their data:** what it does not capture, how it could mislead, and how confident they can be.

Choosing what to measure

A North Star and its inputs

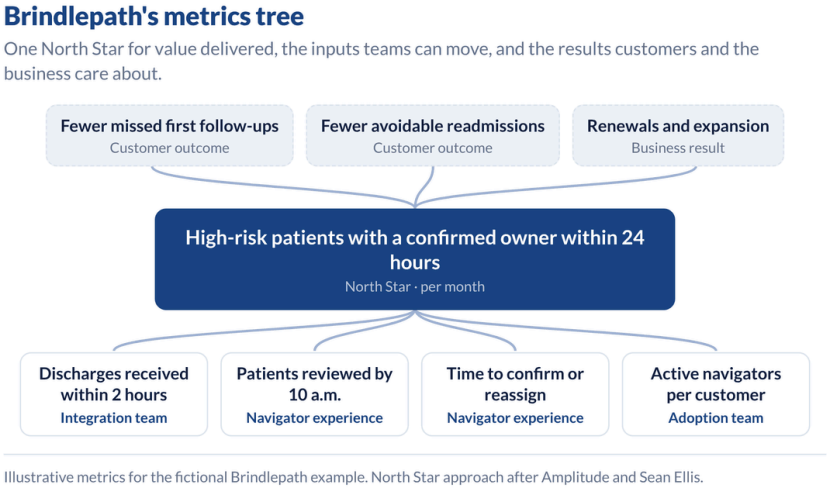
A useful approach, popularized by the analytics company Amplitude and by growth practitioners such as Sean Ellis, is to identify a single **North Star metric**: the measure that best captures the value your product delivers to customers, and that leads to long-term business results.[1] A good North Star:

- reflects value to the customer, not just activity;
- leads revenue, rather than lagging it;
- can be influenced by the product team's work;
- is understandable by everyone in the company.

For Brindlepath, the North Star was **high-risk discharged patients with a confirmed owner within 24 hours**, per month. It captured the core value, a patient someone was looking after, and it led directly to the outcomes cus-

tomers bought the product for: fewer missed follow-ups and fewer avoidable readmissions.

A North Star on its own is too broad to act on. Below it sit a few **input metrics**: the measures teams can move directly, which together drive the North Star. Figure 10.1 shows Brindlepath's metrics tree.



> Figure 10.1. Brindlepath's metrics tree. The North Star captures value delivered; input metrics are what teams can move; customer outcomes and business results sit above and to the side. Each team owns one or two inputs.

Each product team owned one or two inputs. The integration team owned *discharges received within two hours*. The navigator experience team owned *patients reviewed by 10 a.m.* and *time to confirm or reassign*. The adoption team owned *active navigators per customer*. Every team could see how its work connected to the North Star, and from there to the outcomes customers cared about and the revenue the business needed.

Other useful frameworks

Two other frameworks are worth knowing.

Pirate metrics (AARRR), introduced by Dave McClure, follow a customer through Acquisition, Activation, Retention, Referral and Revenue.[2] They are useful for finding the weakest stage in a growth funnel, especially in products with self-service sign-up.

HEART, developed by researchers at Google, measures user experience through Happiness, Engagement, Adoption, Retention and Task success, each paired with goals, signals and metrics.[3] It is especially helpful for measuring the quality of a specific feature or redesign.

Whichever framework you use, the discipline is the same: start from the goal, decide what signal would show progress toward it, and only then choose the metric.

Beware vanity metrics

Vanity metrics go up and to the right and tell you nothing useful: total registered users, cumulative downloads, page views. They are easy to report and hard to act on. The test is simple: if this number changed, would we do anything differently? If not, it is decoration.

Brindlepath's old dashboard tracked *logins per day*, which rose every month. It took a while to notice that much of the increase came from navigators logging in repeatedly because sessions timed out too quickly, which was a usability problem, not a success.

Retention: the number that tells the truth

For subscription products, retention is the most honest measure of value. Customers can be persuaded to buy once; they keep using and paying only if the product keeps helping them.

Cohort analysis shows retention clearly. A cohort is a group of users or customers who started in the same period. Tracking each cohort over time shows whether people keep using the product, and whether recent cohorts retain better than earlier ones, which is the clearest sign that product changes are working.

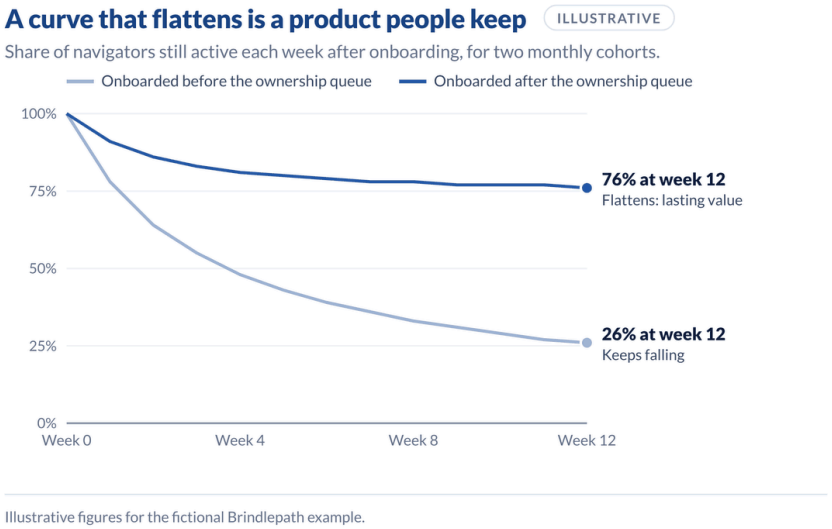


Figure 10.2. Weekly active navigators by monthly cohort (illustrative). The later cohort, onboarded after the ownership queue launched, flattens at a much higher level. A curve that flattens is a product that people keep using.

Look for two things in a retention chart. First, does the curve flatten? A curve that keeps falling toward zero means people are trying the product and leaving. A curve that flattens means a group has found lasting value. Second, are newer cohorts higher than older ones? If so, the product is improving.

In business software, measure retention at two levels: the **user** (are navigators still using it each week?) and the **account** (is the customer renewing and expanding?). Account retention lags user retention by months, so a drop in user engagement is your early warning.

Analyzing data well

Correlation is not causation

The most common analytical mistake is to treat a relationship as a cause. Customers who use the reporting module renew at higher rates. Does reporting cause renewal? Or do larger, more committed customers both use reporting and renew? Without more careful analysis, you cannot tell.

When the decision matters, find stronger evidence: an experiment, a comparison of similar groups, or a before-and-after analysis with a clear control.

Experiments

A/B tests, which randomly show different versions to different users and compare outcomes, are the strongest everyday tool for establishing cause. They work best when you have enough users, when outcomes can be measured quickly, and when showing different versions is acceptable.

In business software, and especially in healthcare, those conditions often do not hold. You may have a few hundred users per customer, outcomes that take months to appear, and clinical workflows where showing different versions to different nurses is inappropriate. Alternatives include staged rollouts across customers or units, with careful before-and-after comparison; pilot and comparison sites; and simple, well-designed observation of real use.

Whatever method you use, decide your measure and your success threshold *before* you look at the results.

Pair numbers with stories

Quantitative data tells you what is happening at scale. Qualitative evidence, from interviews, support tickets and session observations, tells you why. The best insights come from moving between the two. When the numbers show that navigators at one customer review far fewer patients than others, a single call with that customer usually explains why in ten minutes.

Building a measurement foundation

Good analysis depends on good data. A few foundations matter:

- **An event taxonomy:** a consistent, documented set of events and properties that the product records, such as *patient_assigned*, *owner_confirmed* and *patient_reassigned*, with clear definitions.
- **Instrumentation as part of done:** every significant feature ships with the events needed to measure its outcome, as Chapter 9 recommends.

- **One definition of each key metric,** written down, so that "active navigator" means the same thing in every report.
- **Privacy by design:** collect only what you need, avoid sending personal or health information to analytics tools that are not approved for it, and respect customers' data agreements. In healthcare, analytics events should never contain patient identifiers unless the analytics system is covered by the right agreements and controls.
- **Self-service access** for the product team to the data they need, with the data team supporting rather than gatekeeping.

Bringing financial and customer data together

The richest insights often come from joining product usage with financial and customer data. Which features do customers who expand use most? Which usage patterns predict churn six months before renewal? What does it cost to serve customers who use the product heavily versus lightly?

Brindlepath's most useful analysis joined three sources: usage data, support tickets and contract data. It showed that customers whose navigators confirmed ownership for more than 80 percent of high-risk patients in the first 90 days almost always renewed, and those below 50 percent often did not. That single finding reshaped onboarding: the customer success team's first goal became getting every new customer above 80 percent within 90 days.

Brindlepath in practice: finding the leading indicator

Brindlepath's most valuable analysis started with a question from the CEO: could the company tell, months in advance, which customers were likely to renew? Renewal decisions came once a year, far too late to change anything. The team needed a leading indicator.

Maya asked the data team to assemble every customer's history in one place: daily usage by navigator and site, the North Star and input metrics by month, support tickets, implementation milestones and contract outcomes. With only

about forty customers at the time, the analysis was as much careful reading as statistics. The team plotted each customer's ownership rate over its first year, colored by whether it had renewed.

The pattern was striking. Customers whose navigators confirmed ownership for more than 80 percent of high-risk patients within their first 90 days had, almost without exception, renewed. Customers that stayed below 50 percent through the first 90 days had mostly churned or renewed at a reduced scope. The middle group was mixed, and the difference usually came down to whether an executive sponsor had stayed engaged.

The team was careful not to overclaim. With forty customers, the finding was a strong signal, not proof, and it could reflect other differences between customers, such as how well-staffed their navigator teams were. So they treated it as a hypothesis and tested it: for the next ten new customers, customer success made reaching 80 percent within 90 days its first explicit goal, with weekly tracking and a playbook for customers who fell behind. Eight of the ten reached it, faster on average than any earlier cohort, and all eight renewed.

The 90-day ownership rate became the single most-watched number in customer success, appeared on the first page of every account review, and shaped the onboarding work described in Chapter 14. One well-chosen metric, built from joined data and tested before being trusted, did more than ninety-four charts ever had.

Common mistakes

Measuring what is easy instead of what matters. Page views and logins are easy to collect and rarely tell you whether customers got value.

Too many metrics. A dashboard with dozens of charts means nobody knows which ones matter. Fewer, better metrics get looked at.

Deciding the measure after seeing the results. Choosing the metric that makes a launch look good is a common, quiet way of fooling yourself.

Confusing correlation with cause. Customers who use a feature may succeed for reasons that have nothing to do with the feature.

Numbers without stories. Quantitative data shows what is happening; it rarely shows why. A single customer call often explains a puzzling chart.

Leaking sensitive data into analytics. Patient identifiers or other personal data in analytics events create privacy and compliance risk that far outweighs their analytical value.

Try this

Write down your product's North Star metric and three or four input metrics that drive it, in the shape of Figure 10.1. For each input, name the team that owns it and the date it was last reviewed. Then open your main dashboard and remove, or move to a secondary page, every chart that does not connect to the tree. Finally, pick one input metric that moved unexpectedly last month and call two customers to ask what changed for them. Compare what they say with what the numbers suggested.

How AI changes this

Anyone can ask the data a question. AI tools that turn a plain-English question into a database query or a chart put analysis in the hands of everyone on the product team, not just analysts. That is powerful, and risky. The tool will happily answer a question built on the wrong definition of "active." Shared, documented metric definitions, the foundation described above, become more important, not less, because AI tools should use them rather than invent their own.

AI finds patterns quickly. Models can scan usage data for unusual behavior, cluster users by how they use the product, and flag early signs of churn. Treat these findings as leads to investigate, not conclusions. Pattern-finding tools are especially prone to mistaking correlation for cause.

AI summaries need checking. An AI-generated summary of last month's metrics may read beautifully and contain a wrong number. Any AI-produced analysis that will inform a real decision should show its sources and calculations, so a person can check them.

AI features need their own metrics. For AI features, measure quality, not just usage: how often the output is accepted, edited or rejected; how often it is wrong in ways that matter; and what it costs per use. Chapter 17 covers evaluation in detail.

Chapter checklist

- [ ] Do you have a North Star metric that reflects value delivered to customers and leads revenue?
- [ ] Does each team own one or two input metrics that connect to the North Star?
- [ ] Have you removed vanity metrics from your main dashboard?
- [ ] Do you track retention by cohort, for both users and accounts?
- [ ] Do you decide measures and success thresholds before looking at results?
- [ ] Do you use experiments or careful comparisons before claiming a feature caused an outcome?
- [ ] Do you pair quantitative findings with customer conversations?
- [ ] Is there one written definition for each key metric, used by people and AI tools alike?
- [ ] Is instrumentation part of your definition of done, with no personal or health data sent to unapproved tools?
- [ ] Have you joined product usage with financial and customer data to find leading indicators of renewal?

---

1. Amplitude, *The North Star Playbook* (2018); Sean Ellis and Morgan Brown, *Hacking Growth* (Currency, 2017).
2. Dave McClure, "Startup Metrics for Pirates: AARRR!" presentation, 2007.

3. Kerry Rodden, Hilary Hutchinson and Xin Fu, “Measuring the User Experience on a Large Scale: User-Centered Metrics for Web Applications,” *Proceedings of CHI 2010*.

Chapter 11. Trust by Design: Security, Privacy and Compliance

“Security is a process, not a product.”

— Bruce Schneier

The security questionnaire

Brindlepath’s largest deal of the year nearly died in a spreadsheet. The health system’s information security team sent a questionnaire with 312 questions: how data was encrypted, who could access it, how access was reviewed, how incidents were handled, where backups lived, which subcontractors touched patient data, and how the company knew its code was free of known vulnerabilities.

The sales team had expected the questionnaire to take a week. It took seven, because many of the answers did not exist yet. Engineering knew the data was encrypted, but nobody had documented how. Access reviews happened, but informally. The incident response plan was a page on the company wiki that had not been tested. Maya spent her evenings that month answering questions she wished the product had been designed to answer from the start.

The deal closed. Afterward, Maya made a change that outlasted any feature she shipped that year: security, privacy, compliance and accessibility became part of how every product decision was made, not a review at the end.

Customers noticed. The next questionnaire took four days.

Trust is a product feature

The first edition of this book treated security and compliance mostly as regulatory obligations. This edition treats them as the fourth pillar of product leadership from Chapter 1: **trust and responsibility.** Customers buy software they trust. In healthcare, a breach, a privacy failure or a product that excludes people with disabilities is not just a technical failure; it is a failure of the product, and of the people who lead it.

The cost of getting it wrong is high and rising. IBM's annual study of data breaches found that in 2025 the average breach cost $4.44 million across industries, and $7.42 million in healthcare, the most expensive industry for the fourteenth year running. Healthcare breaches also took longer to identify and contain: 279 days on average, against 241 across all industries.[1]

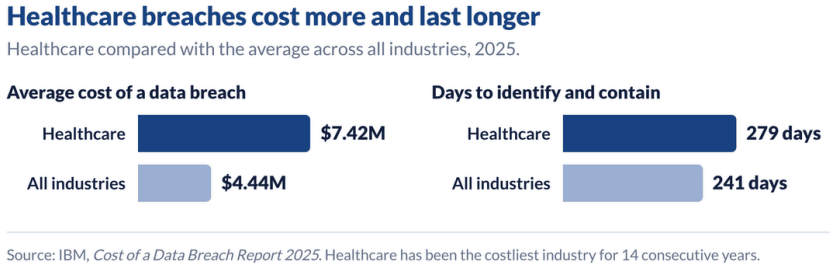


Figure 11.1. Healthcare breaches cost more and last longer than the average. Source: IBM, Cost of a Data Breach Report 2025.

Those numbers measure only the direct costs. They do not capture the deals that never close because a prospect does not trust you, or the customers who quietly leave after an incident.

The landscape of obligations

Every industry has its own rules. The specifics change by country and over time, so treat this section as a map, not legal advice, and work with qualified counsel and compliance professionals.

In US healthcare, a product leader needs to understand at least these:

- **HIPAA** and its Privacy, Security and Breach Notification Rules govern protected health information. Software vendors that handle it on behalf of providers or health plans are typically *business associates* and must sign a **business associate agreement (BAA)** with each customer, and with their own subcontractors that handle the data.
- **Interoperability and information-blocking rules** from the Office of the National Coordinator for Health IT shape how health data must be shared and accessed, and affect products that connect to electronic health records.
- **FDA oversight of software** applies when software functions as a medical device. Some clinical decision support is excluded from device regulation when it meets specific criteria, including that clinicians can independently review the basis for its recommendations. Features that diagnose, treat or drive clinical decisions need careful regulatory analysis early.
- **State privacy laws** increasingly add obligations for health and consumer data beyond HIPAA.
- **Accessibility requirements** are tightening: in 2024, the Department of Health and Human Services finalized a rule requiring organizations that receive HHS funding to make their websites and mobile apps conform to the WCAG 2.1 Level AA standard, with compliance dates phased in from 2026.[2]

Beyond healthcare, most software sold to businesses must show its security practices through independent assessments such as **SOC 2** reports, ISO 27001 certification or, in healthcare, HITRUST. Products sold in Europe must meet the GDPR, and AI features increasingly fall under new rules such as the EU AI Act, whose obligations are phasing in through 2027.

A product leader does not need to be a lawyer. They do need to know which rules apply, which product decisions trigger them, and when to bring in experts.

Security by design

Security by design means building security into every stage of the product's life, rather than testing for it at the end. For a product leader, that means making sure the following happen as a matter of routine.

Threat modeling at design time. For every significant feature, ask: what data does it touch, who should be able to see or change it, and how could it be abused? Fifteen minutes with an engineer and a whiteboard catches problems that weeks of testing would miss.

Least privilege by default. Users, services and AI agents get only the access they need. Brindlepath's navigators see only the patients assigned to their team; support staff can view a customer's configuration but not patient records without an approved, logged request.

Strong identity. Single sign-on with customers' own identity providers, multi-factor authentication, and prompt removal of access when people leave.

Encryption and audit. Data encrypted in transit and at rest, and every access to sensitive records logged in a way customers can review.

Secure development. Code review, dependency scanning, secrets management and regular penetration testing, built into the delivery flow from Chapter 9.

A tested incident plan. Know who does what when something goes wrong, how customers will be told, and how regulators will be notified. Then practice it.

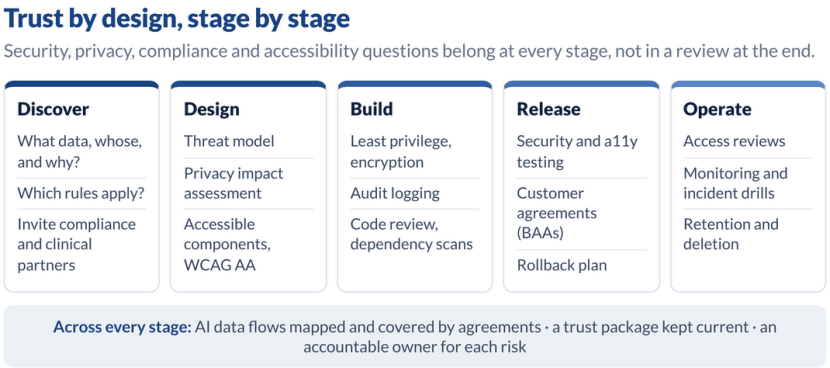


Figure 11.2. Trust by design across the product life cycle. Each stage has its own questions; none of them waits until the end.

Privacy by design

Privacy by design, a set of principles developed by Ann Cavoukian, starts from a simple premise: protect personal information by default, not as an afterthought.[3] For a product team, it comes down to a few habits.

- **Collect the minimum.** Every piece of personal data you collect is a liability as well as an asset. Ask whether you need it before you collect it.
- **Use data only for its stated purpose.** Data collected to coordinate a patient's care should not quietly become training data for an unrelated model.
- **Default to private.** The most protective setting should be the default.
- **Be transparent.** Tell customers, and where relevant patients, what you collect, why, and who you share it with, in plain language.
- **Plan for deletion.** Know how long you keep data, and how you delete or return it at the end of a contract or a product's life, as Chapter 7 discussed.

A **privacy impact assessment** for any feature that uses personal data in a new way makes these habits concrete. It need not be long: what data, for what purpose, with what risks, and what mitigations.

Accessibility is part of quality

Accessible products work for people with disabilities: people who are blind or have low vision, who are deaf or hard of hearing, who have motor or cognitive disabilities, or who rely on assistive technology. In healthcare, those users include patients, family caregivers and many clinicians.

Most software still falls short. WebAIM's 2025 analysis of the home pages of the top one million websites found detectable accessibility failures on 94.8 percent of them, an average of 51 errors per page, most of them common and fixable problems such as low-contrast text and images without text alternatives.[4]

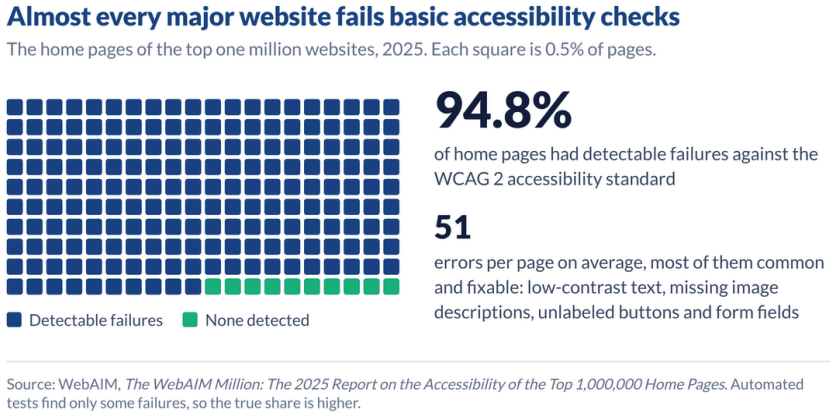


Figure 11.3. Almost every major website has detectable accessibility failures. Source: WebAIM, The WebAIM Million, 2025.

The international standard is the **Web Content Accessibility Guidelines (WCAG)**, published by the World Wide Web Consortium. Level AA of WCAG 2.1 or 2.2 is the common target. For product teams, accessibility is not a separate project. It is part of the definition of done:

- Use a design system whose components are accessible by default.
- Check color contrast, keyboard navigation, screen-reader labels and focus order for every new screen.

- Test with assistive technology and, where possible, with users who rely on it.
- Write in plain language. It helps everyone, including people reading under stress.

Accessibility also makes the product better for everyone. Brindlepath's larger text and keyboard shortcuts, added for navigators with low vision, turned out to be popular with every navigator working a busy afternoon with a headset on.

Working with compliance as a partner

In many companies, compliance is the department that says no at the end of a project. That is usually because it was invited only at the end.

Invite compliance, security and, in healthcare, clinical safety partners into discovery. Share the product brief from Chapter 9 with them. Ask, early, which features will need their review and what they will need to see. Most compliance professionals would far rather shape a design than block a launch.

Build reusable answers. The 312-question security questionnaire taught Brindlepath to maintain a living trust package: security documentation, architecture diagrams, policies, audit reports and standard answers, kept current and shared with prospects early. Some companies publish a public trust center with much of this material. It shortens sales cycles and signals that you take trust seriously.

Brindlepath in practice: the trust package

After the seven-week security questionnaire, Maya asked a product manager to treat the company's security and compliance documentation as a product, with customers (prospects' security teams), a backlog and a release rhythm.

The first version of the trust package took six weeks to assemble. It included an architecture overview showing where patient data lived and how it moved; the company's security policies; the results of its most recent penetration test

and its independent security audit; a description of access controls, encryption, logging and backups; the incident response plan; a list of subprocessors that handled customer data, with their roles; the standard business associate agreement; an accessibility conformance report against WCAG 2.1 Level AA; and a library of answers to the hundred most common questionnaire questions, each with an owner and a review date.

Every new questionnaire became input to the backlog. Questions the package could not answer were added, and the gaps they exposed, such as an access review that had happened informally but was not documented, were fixed in the product or the process rather than papered over in an answer. When Brindlepath began using AI models, the package gained a section describing which models were used, for what, with what data, under what agreements, and with what evaluation and monitoring.

The results showed up in the sales cycle. The average time to complete a security review fell from about seven weeks to under two. Prospects' security teams began telling Brindlepath's sales team that the package was among the clearest they had seen, which, in a market where trust is everything, became a reason to buy. The work cost a fraction of one product manager's time each quarter to maintain.

Common mistakes

Treating compliance as a final gate. Reviews at the end can only approve or block. Involving security, privacy and compliance early lets them shape a better design.

Collecting data just in case. Every field of personal data you store is a liability in a breach. Collect what you need, for a stated purpose.

Assuming the vendor handles it. Cloud providers and AI model providers secure their platforms, not your use of them. Responsibility for how your product handles data stays with you.

Leaving accessibility to a later release. Retrofitting accessibility costs far more than building it in, and excludes users in the meantime.

An incident plan nobody has practiced. The middle of a breach is the worst time to discover that nobody knows who calls the customer.

Overclaiming. Marketing claims about security, compliance or clinical outcomes that the product cannot support create legal and reputational risk.

Try this

Pick your product's most sensitive workflow and walk through it with an engineer, a designer and, if you have one, a security or compliance partner. For each step, ask four questions: What personal data is involved? Who can see or change it, and should they? Is every access logged? Can someone using a screen reader or only a keyboard complete it? Write down every gap you find and add the fixes to your roadmap's capacity for security and compliance. Then schedule a one-hour incident drill for the same workflow.

How AI changes this

AI features raise new trust questions, and product leaders own the answers.

Data flows multiply. Sending patient data to a third-party model provider is a new data flow with its own agreements, risks and obligations. In healthcare, the provider must typically sign a BAA, commit not to retain or train on the data, and meet your security standards. Map every AI data flow as carefully as any other.

New attack surfaces appear. AI systems can be manipulated through their inputs, for example by hidden instructions inside a document an agent reads, a technique known as prompt injection. Agents that can take actions need strict limits on what they can access and do, and human approval for anything consequential.

Outputs can be wrong, biased or unexplainable. A summary that invents a medication, a risk score that disadvantages a group of patients, or a recommendation nobody can explain are trust failures. Evaluate AI features for accu-

racy and fairness before launch and monitor them after, as Chapter 17 describes. Frameworks such as the NIST AI Risk Management Framework offer a structured way to think about these risks.[5]

Transparency matters more. Tell users when content was generated by AI, show the sources behind it where possible, and make it easy for them to correct it. In clinical settings, make clear that AI output supports, and does not replace, professional judgment.

AI can also strengthen trust work. Assistants can draft answers to security questionnaires from your trust package, check new screens for common accessibility failures, review code for security issues, and summarize regulatory changes. As always, an accountable person signs off.

Chapter checklist

- [ ] Do you treat security, privacy, compliance and accessibility as product requirements from the start?
- [ ] Do you know which regulations apply to your product, and which product decisions trigger them?
- [ ] Do significant features get a threat model and, where personal data is used in new ways, a privacy impact assessment?
- [ ] Are least privilege, strong identity, encryption and audit logging defaults in your product?
- [ ] Is there a tested incident response plan, including how customers and regulators will be notified?
- [ ] Is WCAG 2.1 or 2.2 Level AA conformance part of your definition of done?
- [ ] Are compliance, security and clinical safety partners involved in discovery, not just at launch?
- [ ] Do you maintain a current trust package that shortens security reviews?
- [ ] Is every AI data flow mapped, covered by the right agreements, and protected against misuse?
- [ ] Are AI features evaluated for accuracy and fairness, and clearly labelled for users?

1. IBM, *Cost of a Data Breach Report 2025*. Healthcare has had the highest average breach cost of any industry for 14 consecutive years.
2. US Department of Health and Human Services, final rule on nondiscrimination on the basis of disability under Section 504 of the Rehabilitation Act, May 2024. Larger recipients must comply from May 2026, smaller ones from May 2027.
3. Ann Cavoukian, *Privacy by Design: The 7 Foundational Principles* (Information and Privacy Commissioner of Ontario, 2009).
4. WebAIM, *The WebAIM Million: The 2025 Report on the Accessibility of the Top 1,000,000 Home Pages*.
5. National Institute of Standards and Technology, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, January 2023.

PART 4

Customers

CHAPTER 12

Designing with Empathy

“Good design is as little design as possible.”

— Dieter Rams

4:30 in the afternoon

The ownership queue tested beautifully. In usability sessions held in a quiet room at Brindlepath’s office, navigators found their patients in seconds, confirmed ownership in two clicks and rated the experience highly. Three weeks after launch at the first pilot site, usage had fallen by a third.

Lena Park went to find out why. She spent two days sitting beside navigators at the customer, from the start of their shift to the end. By mid-afternoon she understood. The navigators worked in an open-plan office with phones ringing constantly. They used aging computers with small screens, often with a headset on and a patient on the line. The queue’s confirm button was small and sat next to the reassign button; twice that afternoon, Lena watched a navigator reassign a patient by mistake and spend five minutes undoing it. Worse, the queue refreshed when new discharges arrived, so the list jumped under the navigator’s cursor in the middle of a call.

None of that showed up in the usability lab, because the lab was not the navigator’s world. The product worked. It did not work *there*.

The fixes were modest: a larger, separated confirm button, an undo for reassignment, and a queue that announced new arrivals instead of reshuffling. Usage recovered within two weeks and kept rising. The lesson cost Brindlepath a month. It is cheaper to learn it the way Lena eventually did: by going to where the work happens.

Empathy is a method, not a feeling

Empathy in product work is not about being kind or caring, though both help. It is a disciplined practice of understanding the people you build for well enough to anticipate what they need, often before they can say it. It has three parts:

1. **Context:** where, when and under what conditions people use the product.
2. **Goals and emotions:** what they are trying to achieve, what they fear, and what success feels like to them.
3. **Constraints:** what limits them: time, training, technology, rules, other people.

In healthcare, the stakes of getting this wrong are high, and the people you design for are often exhausted. Clinicians already spend a large share of their day on screens. In a widely cited time-and-motion study of 57 physicians in four specialties, Christine Sinsky and colleagues found that only 27 percent of the office day went to direct clinical face time with patients, while 49 percent went to the electronic health record and desk work: nearly two hours of screen work for every hour with patients.[1]

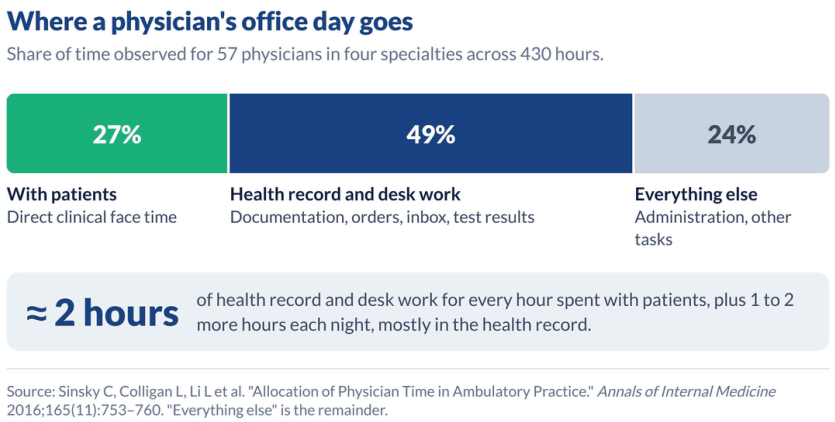


Figure 12.1. Where a physician's office day goes. For every hour with patients, nearly two more go to the health record and desk work. Every product that adds a click adds to that burden. Source: Sinsky et al., Annals of Internal Medicine, 2016.

Every product sold into that environment either adds to that burden or relieves it. Empathy is how you know which.

Methods for understanding people

Contextual inquiry. Watch people do their real work in their real environment, and ask them to explain as they go. This is the single most powerful empathy method, and the one most often skipped because it takes time and travel. Lena's two days beside navigators found more than a month of lab testing.

Shadowing. Follow one person through a whole shift or day. You see the transitions, interruptions and workarounds that no interview reveals.

Journey mapping. Map the steps a person goes through to achieve a goal, including what they do, think and feel at each step, and where your product touches the journey. Journey maps make pain points visible to the whole team, including people who never meet customers.

Diary studies. Ask participants to record their experiences over days or weeks. Useful for understanding behavior that unfolds over time, such as a patient's recovery after discharge.

Personas, used carefully. A persona is a short description of a type of user, based on research. Personas help teams remember who they are building for. They become harmful when they are invented, or when they replace contact with real people. A persona should be a summary of evidence, revisited when the evidence changes.

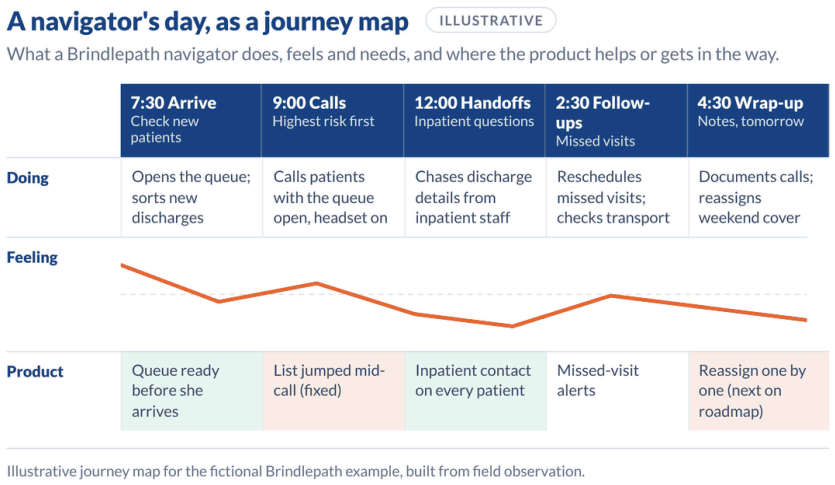


Figure 12.2. A navigator's day as a journey map (illustrative). The emotion line shows where the work is hardest; the bottom row shows where Brindlepath helps and where it was getting in the way.

Designing for the real environment

Lena's discovery generalizes into a few principles for products used under pressure.

Design for interruption. People will be pulled away mid-task. Save state automatically, make it obvious where they left off, and never lose work.

Prevent errors, and make them easy to undo. Separate destructive actions from common ones, confirm only when an action is truly hard to reverse, and offer undo everywhere else. Confirmation dialogs that appear constantly train people to click through them.

Keep the screen still. Do not move things under someone's cursor. Announce changes; let the user choose when to see them.

Minimize clicks and typing for frequent tasks. A task done forty times a day deserves to be fast. Use keyboard shortcuts, sensible defaults and bulk actions.

Use plain language. Write for someone who is tired and distracted. Labels should say what happens, not what the system calls it.

Design for the hardware and the setting. Old screens, shared computers, gloves, glare, noise. Test on the actual devices in the actual place.

Design for everyone. The accessibility practices in Chapter 11 make products better for people with disabilities and for everyone working under strain.

The users who never log in

In healthcare, the person who benefits most from your product often never sees it. Brindlepath's real beneficiary was the patient who went home from the hospital and needed someone to notice if things went wrong. That patient never logged in, but the product's design affected them directly: how quickly a navigator called, whether the navigator had the right information, whether the follow-up visit happened.

Designing with empathy for the beneficiary means asking, at every decision, how it changes the experience of the person at the end of the chain. Brindlepath's team began each design review with a short story of a real (anonymized) patient whose care depended on the workflow in question. It changed conversations. "This adds a step for navigators" became "this adds a step, and here is what it means for Mrs. Alvarez's follow-up call."

The same principle applies in other industries: the student behind the teacher's dashboard, the customer behind the support agent's screen, the citizen behind the government caseworker's system.

Balancing empathy and innovation

Empathy keeps products grounded in real needs. Taken too literally, it can also keep them timid. Users describe their problems in terms of what they know, and their suggestions tend to be small improvements to today's workflow. Breakthrough products often change the workflow itself.

The answer is not to ignore users, but to separate their problems, which you should trust deeply, from their proposed solutions, which you should treat as one input among many. Brindlepath's navigators asked for a better spreadsheet. What they needed was for ownership to be assigned before they arrived in the morning, which no spreadsheet could do. The insight came from empathy; the solution came from the team's knowledge of what technology could do.

A few practices help balance the two:

- **Test bold ideas with low-fidelity prototypes,** so users react to the concept rather than the polish.
- **Introduce change gradually** where workflows are safety-critical, with the option to use the old way during transition.
- **Measure whether the innovation actually relieves the burden,** using time-on-task, error rates and the users' own sense of workload.
- **Involve users as partners,** not just test subjects. Brindlepath formed a small advisory group of navigators from three customers who saw designs early and helped shape them.

Continuous improvement for user-centricity

Empathy is not a phase. People's work changes, products change, and the gap between them opens again. Keep it closed with a few lasting habits:

- **Regular field visits** for every product trio, at least once a quarter.
- **A steady stream of feedback** from support tickets, in-product feedback and customer success, reviewed by the product team weekly.

- **Usability measures** for key tasks, tracked over time: completion rates, time on task, error rates.
- **Satisfaction measures** used with care. Net Promoter Score and similar surveys can show trends, but a score without the stories behind it says little about what to fix.

Brindlepath in practice: the navigator advisory group

Lena's discovery in the opening scene led to a lasting practice. Brindlepath formed a navigator advisory group: eight navigators and two navigator managers from four customers, who met with the product team for an hour every month and were paid a modest stipend, with their employers' agreement, for their time.

The group saw designs early, usually as sketches or rough prototypes, before the team had invested much in any direction. They tested new workflows in their own environments for a week before wider release. And they brought problems the product team had not heard about: the navigator who covered a rural clinic with an unreliable connection, the manager who needed to rebalance workloads when someone called in sick, the new hire who could not tell which patients were most urgent on her first day.

Two rules kept the group useful. First, the product team presented problems and options, not finished solutions, and asked the group to argue about them. Second, the team closed the loop: every month began with a short review of what had changed because of the group's input in previous months. Members said this was the reason they kept coming. Seeing their suggestions in the product made them advocates inside their own organizations, which customer success noticed in renewal conversations.

The group did not replace field visits or broader research. Ten people cannot represent every navigator. But it gave the product team a standing relationship with real users, and it caught several problems, including a confusing change to how risk levels were displayed, before they reached hundreds of navigators.

Accessibility in a real workflow

When Brindlepath ran its first full accessibility review of the ownership queue, it found problems no usability session had surfaced. The risk level was shown only by color, so a navigator with color blindness could not tell high from medium risk. The queue could not be used with a keyboard alone. A screen reader announced each patient row as “button, button, button.”

None of these was hard to fix. Risk levels gained text labels and icons. Every action gained a keyboard shortcut and a visible focus state. Rows were labelled with the patient’s name, risk level and discharge time. The fixes took one engineer two weeks, and the team added the checks to its definition of done so new screens would not repeat the mistakes. The keyboard shortcuts became one of the most popular features in the product, among navigators with and without disabilities.

Common mistakes

Testing only in the lab. Quiet rooms, fast computers and focused participants hide the interruptions, noise and fatigue of real work.

Inventing personas. Personas made up in a workshop reflect the team’s assumptions. Build them from research and revisit them when the evidence changes.

Designing only for the logged-in user. Patients, students, citizens and customers who never see the product are often the people it most affects.

Taking users’ solutions literally. Users are experts in their problems. Their proposed solutions are one input, often a modest improvement on today’s workaround.

Adding confirmation dialogs everywhere. Constant confirmations train people to click through them. Prevent errors by design and offer undo instead.

Equating empathy with satisfaction scores. A score shows a trend; it rarely tells you what to fix. Stories and observation do.

Try this

Arrange for every member of your product trio to spend half a day with a user in their real environment within the next month, watching rather than demonstrating. Each person should come back with three observations: one workaround, one moment of frustration and one moment where the product helped. Combine them into a simple journey map like Figure 12.2. Then pick the most painful moment on the map and make it the subject of your next round of discovery.

How AI changes this

AI can reduce burden, or add to it. The most promising AI features in healthcare, such as ambient documentation that drafts clinical notes from a conversation, aim directly at the screen time in Figure 12.1. But an AI feature that produces output clinicians must carefully check adds burden of a different kind. Design AI features around the reviewer's time: make outputs easy to verify, highlight uncertainty, and make correction fast.

AI can scale empathy research. Assistants can transcribe and tag hours of field recordings, cluster support tickets by emotion and theme, and draft journey maps from research notes. That leaves more time for the field visits themselves, which no tool replaces.

Synthetic users are not users. As Chapter 3 warned, AI-generated personas can rehearse an interview guide, but they cannot tell you what it is like to be a navigator at 4:30 in the afternoon. Lena's lesson does not change.

AI interfaces need empathy too. Conversational and agent-based features raise new design questions: how much the AI should do on its own, how to show what it did and why, how to let people stay in control, and how to avoid sounding more certain than it is. Chapter 17 covers these in depth.

Chapter checklist

- [ ] Has every member of your product trio watched real users work in their real environment in the last quarter?
- [ ] Do you have a journey map for your most important workflow, built from field research?
- [ ] Are your personas summaries of evidence, revisited when the evidence changes?
- [ ] Does your product save work through interruptions, prevent errors and make them easy to undo?
- [ ] Do you test on the devices and in the settings where the product is actually used?
- [ ] Do design reviews consider the people who benefit from the product but never log in?
- [ ] Do you separate users' problems, which you trust, from their proposed solutions?
- [ ] Do you track usability measures for key tasks over time?
- [ ] Are AI features designed around the time it takes a person to check and correct them?

---

1. Christine Sinsky, Lacey Colligan, Ling Li et al., "Allocation of Physician Time in Ambulatory Practice: A Time and Motion Study in 4 Specialties," *Annals of Internal Medicine* 165, no. 11 (2016): 753–760. The study observed 57 physicians for 430 hours.

Chapter 13. Go-to-Market and Launch

“Marketing is not the art of finding clever ways to dispose of what you make. It is the art of creating genuine customer value.”

— Philip Kotler

The launch nobody noticed

Brindlepath’s first big release after the new strategy was the outcome report: a view that showed a health system’s chief nursing officer and chief financial officer how many high-risk patients had been owned within 24 hours, how many follow-ups had been kept, and what that was likely worth in avoided readmissions. The product team was proud of it. It shipped on time, worked well, and was announced in a release note and a customer newsletter.

Three months later, four customers had opened it. The sales team had not mentioned it in a single deal. Customer success managers had not shown it in a single business review.

The report was not the problem. The launch was. Nobody outside the product team understood who the report was for, what it let them do, or how to use it in a conversation with a customer. Maya relaunched it properly: a one-page story for each audience, training for sales and customer success, a short demo built around a real (anonymized) customer’s numbers, and a target to show it

in every quarterly review. Within a quarter, it was the most-requested part of every sales demo, and three customers used it to justify expanding to new hospitals.

The lesson is old, and every product team relearns it: building the product is half the job. Getting it into the hands and heads of the people it is for is the other half.

What go-to-market means

A **go-to-market (GTM) strategy** is the plan for how a product reaches its customers and wins their business: who you sell to, what you say, through which channels, at what price, and how you support them after the sale. It connects the strategy and positioning from Chapter 4, the pricing from Chapter 6 and the customer success work in Chapter 14.

A complete GTM plan answers six questions:

1. **Who exactly is the target?** The best-fit customer profile and the buying group from Chapter 2.
2. **What is the message?** The positioning, translated into language for each role in the buying group.
3. **How will they find and buy it?** The motion and channels: direct sales, self-service, partners, marketplaces.
4. **What will it cost them, and how will they buy?** Pricing, packaging, contract terms, procurement path.
5. **How will they succeed with it?** Onboarding, implementation, training, support.
6. **How will we know it is working?** Pipeline, win rates, sales cycle length, time to value, early adoption.

Choosing a motion

Software reaches customers in a few broad ways, and the choice shapes the product itself.

- **Sales-led.** A sales team finds, qualifies and closes deals, often with long cycles and many stakeholders. Typical for large, complex, regulated purchases such as Brindlepath's.
- **Product-led.** Users discover, try and adopt the product themselves, often through a free tier or trial, and expand over time. Typical for tools that individuals or small teams can adopt without procurement.
- **Partner-led.** Resellers, system integrators, platforms or marketplaces carry the product to customers. Common when a partner already owns the relationship or the channel, as Chapter 15 discusses.

Three ways to go to market

Most business software ends up with a blend. The highlighted column is Brindlepath's main motion.

	Sales-led	Product-led	Partner-led
Who decides	A buying group of executives, IT and clinical leaders	An individual or small team	The partner's customer, guided by the partner
How they start	Demos, a priced pilot, a contract	Free trial or free tier	Bundled in a partner's offer or marketplace
Sales cycle	Months, sometimes a year or more	Minutes to weeks	Depends on the partner
Product must	Meet security, integration and procurement needs	Deliver value with no help from you	Fit the partner's platform and workflow
Main measures	Win rate, cycle length, pilot conversion	Activation, conversion to paid	Partner-sourced pipeline, attach rate
Watch out for	High cost of sale; custom promises	Low-value users; hitting a ceiling	Losing the customer relationship

> Figure 13.1. Go-to-market motions compared. Most business software ends up with a blend: product-led adoption inside accounts, sales-led expansion across them, and partners for reach.

Most companies end up with a blend. Brindlepath was firmly sales-led, but it learned from product-led companies: it made the pilot experience as self-explanatory as possible, so navigators adopted the product with little training, and their enthusiasm helped the sales team expand to other hospitals in the same health system.

Messaging for each audience

Positioning, from Chapter 4, is the shared understanding of what you are and why you matter. **Messaging** is how you express it to each audience, in their words, about their concerns.

For each role in the buying group, write three things: the problem they feel, the outcome they want, and the proof they need. For Brindlepath's outcome report:

Audience	Problem they feel	Outcome they want	Proof they need
Chief nursing officer	Can't show the board that care transitions are improving	A clear, credible view of progress	Their own data, trended over time
Chief financial officer	Readmission penalties with no clear plan to reduce them	A return on the investment	Avoided readmissions, valued conservatively
Navigator manager	Staff overloaded, no way to see where help is needed	Balanced workload, fewer missed patients	Patients per navigator, missed follow-ups by team

Test messages with real buyers before launch. Read them aloud in a customer's voice. If a sentence sounds like a brochure, rewrite it until it sounds like something a customer would say to a colleague.

Building the launch plan

Not every release deserves the same launch. A useful practice is to **tier launches** by their importance to customers and the business, and match the effort to the tier.

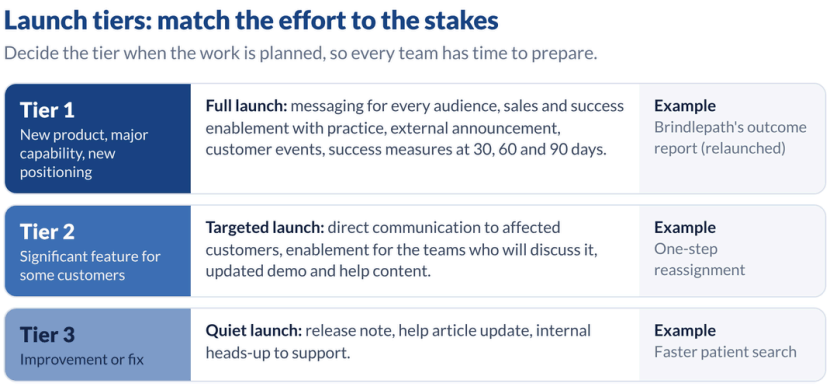


Figure 13.2. Launch tiers. Match the launch effort to what is at stake, and decide the tier early, when the work is planned, not the week before release.

A **Tier 1** launch, a new product, a major capability or a change that shifts the company's positioning, gets the full treatment: messaging for each audience, sales and customer success enablement, external announcement, customer events, and measures of success. A **Tier 2** launch, a significant feature, gets targeted communication to affected customers and enablement for the teams that will discuss it. A **Tier 3** launch, an improvement or fix, gets a release note and an update to help content.

Decide the tier when the work is planned, not the week before release, so marketing, sales and customer success have time to prepare.

Enabling sales and customer success

The sales and customer success teams are the product's voice with customers. If they cannot explain a capability confidently, it effectively does not exist. Good enablement includes:

- **The story:** a short narrative of the problem, the solution and the outcome, with a customer example.
- **The demo:** a script built around a realistic scenario, not a tour of every button.

- **The objections:** the hard questions customers will ask, with honest answers.
- **The boundaries:** what the capability does not do, so nobody over-promises.
- **Practice:** time for people to rehearse and ask questions, not just a slide deck in their inbox.

Pilots that become purchases

In business software, and particularly in healthcare, the path to a sale often runs through a pilot. Chapter 3 described pilots designed to end in a decision: agreed measures, a baseline, a success threshold, a fixed duration and a named decision-maker. From a go-to-market perspective, three more practices matter.

Price the pilot. Even a modest fee changes behavior. Customers who pay for a pilot assign people to it and take the decision seriously.

Design the pilot for expansion. Choose a scope that, if it succeeds, naturally leads to a larger rollout: one hospital in a system of five, one service line in a hospital.

Write the expansion terms in advance. Agree what happens if the success threshold is met: the price, scope and timeline of the full deployment. Then the decision at the end of the pilot is about results, not a fresh negotiation.

In the Brindlepath story, the share of pilots that became contracts rose from under half to about three in four once the company adopted these practices, and the time from pilot start to signed contract fell by several months. Your own results will depend on your market, but the direction is one I have seen repeatedly.

Launching in regulated markets

Launching in healthcare and other regulated markets adds steps a product leader must plan for:

- **Customer approvals.** Health systems may require clinical, security, legal or IT review before a new capability can be turned on, sometimes for each hospital.
- **Training and change management.** A change to a clinical workflow may need formal training and updated procedures at the customer. Feature flags, from Chapter 9, let you enable capabilities customer by customer when they are ready.
- **Claims and evidence.** Be careful what you claim. Statements about clinical outcomes or cost savings should be supported by evidence and reviewed by legal and compliance. Use customers' own results, with permission, rather than broad promises.
- **Contract and procurement timelines.** Budget cycles, procurement committees and contract reviews can add months. Know your customers' fiscal calendars and plan launches and pilots around them.

After the launch

A launch is a hypothesis about adoption. Check it.

Set measures before launch: how many customers will adopt the capability within 30, 60 and 90 days; how often it will appear in sales conversations; what share of eligible users will use it. Review them on schedule, and talk to customers who did not adopt to find out why. Sometimes the product needs work. Often, as with Brindlepath's outcome report, the launch did.

Brindlepath in practice: relaunching the outcome report

The relaunch of the outcome report, described at the start of this chapter, followed the Tier 1 playbook.

The product marketing lead and Maya wrote a one-page story for each audience. For chief nursing officers: "Show your board that care transitions are improving, with your own data." For CFOs: "See the value of avoided readmis-

sions, calculated conservatively, with the assumptions shown.” For navigator managers: “See where your team needs help before patients are missed.”

The demo was rebuilt around a realistic scenario. A health system with five hospitals, one of them lagging, and a chief nursing officer preparing for a board meeting. The demo took eight minutes, and it ended with the question every buyer was actually asking: what is this worth to us?

The enablement session ran for two hours rather than the usual thirty minutes. Salespeople and customer success managers each demoed the report to a colleague playing a skeptical CFO, and the hardest questions were collected into an objections guide. The most common question, “How do you calculate avoided readmissions?”, got a careful, honest answer: a conservative method, the assumptions behind it, and a reminder that the customer’s own finance team should validate the figure.

Customer success set a goal to show the report in every quarterly review for the next two quarters, and tracked it. Three measures were reviewed at 30, 60 and 90 days: customers who opened the report, appearances in sales opportunities, and expansions where the report was cited.

At 90 days, the report had been opened by most customers, appeared in most active sales opportunities, and had been cited by three customers as part of the case for expanding to new hospitals. The product had not changed at all between the first launch and the relaunch. Only the launch had.

Pricing the pilot, in practice

Brindlepath’s early pilots were free, on the theory that removing cost would remove friction. It did. It also removed commitment. Customers assigned whoever was available, missed weekly reviews and let pilots drift.

The company began charging a modest fee for every pilot, credited against the first year’s contract if the customer went ahead. The effect on behavior was immediate. Customers assigned a named manager, protected navigators’ time

for training, and came to decision meetings prepared. The fee was small enough that it rarely slowed a deal, and the few prospects who would not pay anything turned out to be the ones least likely to buy.

Common mistakes

Launching to the product team. A release note and a newsletter mention reach the people who already know about the feature. Customers and sellers need a story, a demo and a reason to care.

One message for everyone. The navigator, the chief nursing officer and the CFO care about different things. A single message usually speaks to none of them.

Enabling sales with slides. Sellers learn by practicing, not by reading. Give them a demo script, the hard questions and time to rehearse.

Free pilots with no exit. Unpriced pilots without expansion terms invite casual commitment and end in renegotiation rather than a decision.

Forgetting the customer's calendar. Health systems and other large buyers have budget cycles, approval committees and training windows. Launches that ignore them stall.

Overpromising AI. Claims an AI feature cannot consistently meet damage trust far more than the feature builds it.

Try this

Pick your next significant release and write the messaging table from this chapter for it: each role in the buying group, the problem they feel, the outcome they want and the proof they need. Then give a salesperson ten minutes to demo the capability to you, cold, using only the materials you have prepared. Note every question they cannot answer and every claim they make that the product does not support. That list is your enablement plan.

How AI changes this

AI accelerates launch work. Assistants can draft messaging variants for each audience, release notes, help articles, sales talk tracks and training quizzes from the product brief. They can analyze win and loss notes and call transcripts to find which messages resonate. Review everything for accuracy, especially claims.

Launching AI features needs extra care. Customers, especially in healthcare, want to know how an AI feature works, what data it uses, how accurate it is, and what happens when it is wrong. Prepare clear answers, documentation of evaluation results, and guidance on appropriate use. Over-promising on an AI feature damages trust faster than almost anything else.

Buyers will ask about your AI posture. Even for non-AI features, many security and procurement reviews now include questions about how a vendor uses AI, where customer data goes, and whether it is used to train models. Add these answers to the trust package from Chapter 11.

AI changes how buyers research. Buyers increasingly ask AI assistants to compare vendors and summarize options before they talk to sales. Clear, specific, well-structured public information about what your product does and for whom helps you show up accurately in those answers.

Chapter checklist

- [ ] Does your GTM plan answer who, what message, which channels, what price, how customers succeed, and how you will measure it?
- [ ] Have you chosen a motion, or blend of motions, that fits how your customers buy?
- [ ] Do you have messaging for each role in the buying group, tested with real buyers?
- [ ] Do you tier launches, and decide the tier when work is planned?
- [ ] Do sales and customer success get the story, demo, objections, boundaries and practice before launch?

- [ ] Are pilots priced, designed for expansion, and backed by agreed expansion terms?
- [ ] Have you planned for customer approvals, training, evidence for claims, and procurement timelines?
- [ ] Do you measure adoption after launch, and talk to customers who did not adopt?
- [ ] Are you ready to explain how your AI features work, how accurate they are, and how they use data?

Chapter 14. Customer Success and Retention

“Your most unhappy customers are your greatest source of learning.”

— Bill Gates

The renewal that almost wasn’t

Ninety days before Ambermoor Health’s renewal, Brindlepath’s customer success manager, Tomás Rivera, ran his usual review and found something alarming. Ambermoor was Brindlepath’s largest customer, and on paper everything was fine: no open escalations, a friendly relationship with the director of care management, and an invoice paid on time. But the usage data told a different story. At two of Ambermoor’s five hospitals, the share of high-risk patients with a confirmed owner within 24 hours had fallen from over 80 percent to under 40 percent in three months.

Tomás called the navigator managers at both hospitals. One had lost three experienced navigators to retirement, and the new hires had never been trained on the product. The other had changed how discharges were coded, so a third of high-risk patients no longer appeared in the queue at all.

Neither problem would have reached Brindlepath through a support ticket. Both would have shown up at renewal as a vague sense that “the product isn’t working for us anymore.” Instead, Brindlepath trained the new navigators,

fixed the coding mapping with the hospital's IT team, and walked into the renewal conversation with a chart showing both hospitals back above 80 percent. Ambermoor renewed, and added two more hospitals.

Customer success is the discipline of noticing those signals early, and acting before the customer has to ask.

Customer success is not support

Support responds when something is broken. **Customer success** works proactively to make sure customers achieve the outcomes they bought the product for, and keep achieving them. Support is reactive and measured by how quickly problems are resolved. Customer success is proactive and measured by retention, expansion and, above all, customer outcomes.

The distinction matters because subscription businesses live or die on retention. A customer who signs a contract has bought the right to decide again at renewal. Every month in between, the product either proves its value or erodes it.

Why retention compounds

Small differences in retention produce large differences over time. Figure 14.1 follows the same $10 million of annual recurring revenue over five years at three different rates of **net revenue retention**: the revenue you keep from existing customers, including expansion, as introduced in Chapter 6.

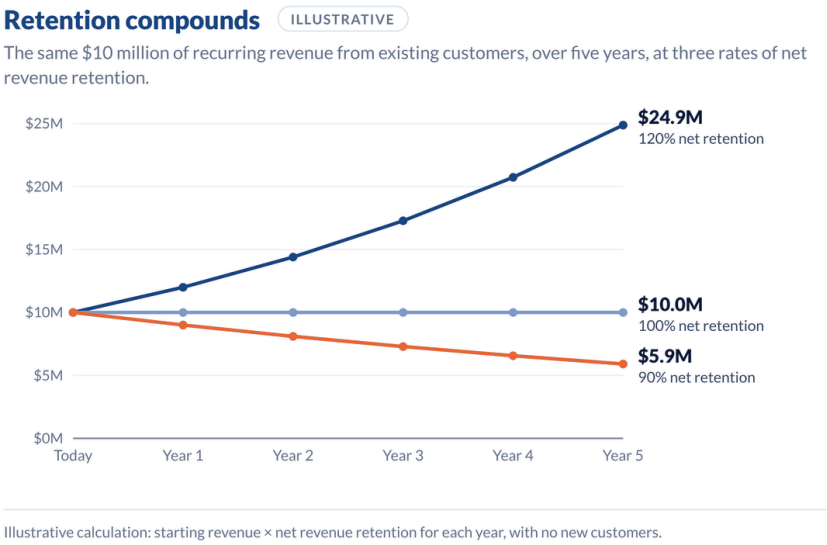


Figure 14.1. The same $10 million of recurring revenue after five years, at three rates of net revenue retention (illustrative). Small differences in retention compound into very different businesses.

At 90 percent net retention, the base shrinks to about $5.9 million in five years, and the company must win more than $4 million of new business just to stand still. At 100 percent, it holds steady. At 120 percent, existing customers alone grow it to nearly $25 million. That is why the best software businesses invest heavily in the customers they already have: retention and expansion are usually cheaper than acquisition, and they compound.

The customer life cycle

Customer success follows customers through a life cycle, with different work at each stage.

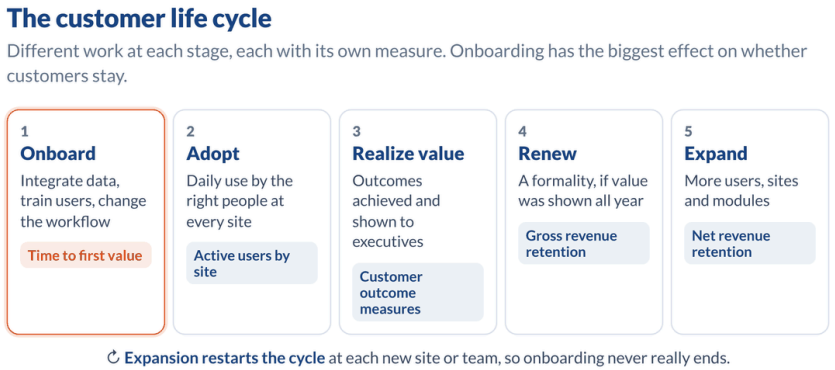


Figure 14.2. The customer life cycle, with the main measure for each stage. The most important number is time to first value: how quickly a new customer sees the outcome they bought.

Onboarding and implementation. The customer sets up the product, connects data and trains users. In healthcare, this often includes record integration, security review and clinical workflow changes. The goal is **time to first value**: how quickly the customer sees the outcome they bought. For Brindlepath, first value was the first week in which a hospital's navigators confirmed ownership of more than 80 percent of high-risk patients.

Adoption. Users build the product into their daily work. Customer success tracks usage, finds teams or sites that are lagging, and helps them catch up.

Value realization. The customer achieves and can demonstrate the outcomes they bought: fewer missed follow-ups, fewer readmissions, saved time. Regular business reviews make this value visible to the executives who will decide on renewal.

Renewal. The customer decides to continue. Renewals should be a formality by the time they arrive, because the value has been shown throughout the year.

Expansion. The customer extends the product to more users, sites or modules. Expansion is the natural result of proven value, not a separate sales campaign.

Onboarding matters most

Of all these stages, onboarding has the largest effect on long-term retention. Customers who reach first value quickly tend to stay; customers who struggle through a long implementation often never fully recover their enthusiasm. Brindlepath's analysis in Chapter 10 found that customers who reached 80 percent ownership within 90 days almost always renewed.

To shorten time to first value:

- **Define first value precisely,** for each type of customer, and track it.
- **Remove setup work from the customer.** Standard integrations, templates and sensible defaults beat custom configuration.
- **Start small.** Get one team or site to value first, then expand.
- **Assign clear owners** on both sides, with a shared plan and dates.
- **Celebrate first value** with the customer's executives, so they see it early.

Health scores: seeing trouble early

A **customer health score** combines several signals into an early warning of risk or opportunity. There is no universal formula, and simpler is usually better. A good health score includes:

- **Outcome measures:** is the customer achieving what they bought? For Brindlepath, the ownership rate and follow-up rate at each site.
- **Adoption measures:** are the expected users active? Is usage growing or falling?
- **Relationship signals:** is there an engaged executive sponsor? Has the champion left?
- **Support signals:** are there unresolved issues or a rising trend in tickets?
- **Commercial signals:** are invoices paid on time? Has the customer asked about price or contract changes?

Weight outcome and adoption signals most heavily. Relationship signals matter, but a friendly contact can hide a failing deployment, as Tomás found at Ambermoor. Review health scores regularly and, more important, act on them: each at-risk customer should have an owner and a plan.

Turning feedback into product

Customer success teams hear more from customers than anyone else in the company. That knowledge belongs in the product.

Create a regular channel. Brindlepath's customer success managers met the product trios every two weeks to share patterns: what customers struggled with, what they asked for, where they found workarounds.

Bring evidence, not just requests. "Three customers asked for bulk reassignment" is useful. "At three customers, navigators spend about 20 minutes each Monday reassigning weekend discharges one by one" is far more useful, because it describes the problem.

Close the loop. When customer feedback shapes the product, tell the customers who raised it. Few things build loyalty faster than seeing your suggestion in a release note.

Advocacy

Satisfied customers are your most credible marketing. In business software, and especially in healthcare, where buyers trust their peers far more than vendors, a reference call from a respected customer can decide a deal.

Build advocacy deliberately:

- **Help customers tell their story,** in case studies, conference talks and peer conversations, with their outcomes and in their words.
- **Create a customer community,** such as user groups or an advisory board, where customers learn from each other and from you.
- **Respect your advocates' time.** Do not overuse a small group of references. Reward them with early access, influence on the roadmap and

recognition.

- **Get permission and stay accurate.** Any claim about a customer's results should be approved by the customer and supported by data.

Brindlepath in practice: the quarterly business review

Brindlepath's customer success team rebuilt its quarterly business reviews around one principle: the customer's outcomes come first, the product second.

Each review followed the same structure and lasted under an hour. It opened with the customer's own goals, restated from the start of the relationship: for Ambermoor, reducing missed follow-ups and avoidable readmissions across its hospitals. Then came the outcome measures, trended over the quarter and the year, by hospital: the ownership rate, missed first follow-ups and the estimate of avoided readmissions from the outcome report. Then adoption: active navigators by site, with any site falling behind named openly and a plan to help. Then what the customer had told Brindlepath, and what Brindlepath had done about it. Only then the roadmap: what was coming that would help this customer specifically, and what the product team wanted to learn from them. The review closed with agreed next steps, each with an owner on both sides.

Two details made the difference. Tomás and his colleagues sent the outcome data to the customer's champion a week ahead, so there were no surprises in front of executives. And they always invited the executive sponsor, not just the day-to-day users. If the sponsor could not attend, they rescheduled. A review without the person who decides on renewal is a status meeting.

Over two years, the reviews became one of the most valued parts of the relationship. Several customers began using the same outcome slides in their own board meetings, which was the clearest possible sign that Brindlepath had made its value visible.

Onboarding, redesigned

Before the 90-day finding in Chapter 10, Brindlepath's onboarding was a sequence of technical steps: contract signed, integration built, users trained, live. Time to first value averaged more than five months, mostly spent waiting on integration work and approvals.

The redesign reorganized onboarding around first value. Integration moved onto the standard connector, which cut the technical work from months to weeks. One hospital and one navigator team went live first, rather than the whole health system at once. The customer's navigator manager and Brindlepath's onboarding lead shared a written plan with dates and a single goal: 80 percent ownership within 90 days of go-live at the first hospital. Training became short, role-specific sessions in the navigators' own environment, followed by a check-in after the first week.

Average time to first value fell to under three months, and the share of new customers reaching 80 percent within 90 days rose sharply. Expansion to the remaining hospitals in each system began sooner, because the first hospital's results made the case internally.

Common mistakes

Mistaking friendliness for health. A warm relationship with one contact can hide a failing deployment. Watch outcomes and usage, not just sentiment.

Ignoring onboarding. Customers who struggle through a long implementation often never fully recover their enthusiasm, whatever you do later.

Measuring support, not success. Ticket resolution time matters, but it does not tell you whether customers are achieving what they bought.

Treating renewal as a sales event. If value has not been shown all year, a renewal meeting is too late to start.

Burning out your advocates. The same three references asked for every deal soon stop answering. Grow your advocate community and reward the people in it.

Keeping customer knowledge in customer success. Patterns that customer success managers see every week belong in product decisions. Without a regular channel, they stay in their heads.

Try this

Pick your five largest customers. For each, write down their outcome measure, its trend over the last quarter, the number of active users by site, the name of their executive sponsor and the date of your last conversation with that sponsor. Mark any customer where the outcome or usage is falling, or where you are unsure the sponsor is still in place. Each marked customer gets an owner and a plan this week, before the renewal conversation makes the decision for you.

How AI changes this

AI can watch every account. An AI assistant can monitor usage, support tickets, emails and meeting notes across all customers and flag the kind of decline Tomás found at Ambermoor, without waiting for a quarterly review. It can draft the account summary before every call. Treat its alerts as prompts for a human conversation, not as conclusions.

AI can deliver help at the moment of need. In-product assistants that answer questions using your help content, guide users through tasks and explain what changed in a release can shorten onboarding and reduce support load. Make sure they draw only on accurate, current content, and hand off to a person when they cannot help.

AI can scale customer success. With AI handling research, preparation and routine follow-up, a customer success manager can serve more accounts with more attention, rather than less. The relationship itself, the trust built through honest conversations, stays human.

AI features need their own success plan. Customers adopting AI features often need more guidance, not less: what the feature is good at, where to double-check it, and how to give feedback when it is wrong. Build this into

onboarding.

Chapter checklist

- [ ] Do you distinguish customer success, which is proactive and outcome-focused, from support?
- [ ] Do you track net and gross revenue retention, and understand how they compound?
- [ ] Have you defined time to first value for each type of customer, and do you track it?
- [ ] Does your health score weight outcomes and adoption most heavily, and does every at-risk account have an owner and a plan?
- [ ] Do customer success and product teams meet regularly to share evidence about customer problems?
- [ ] Do you close the loop with customers whose feedback shapes the product?
- [ ] Do you build advocacy deliberately, with customers' permission and accurate claims?
- [ ] Are AI-driven alerts treated as prompts for human conversations?
- [ ] Does onboarding for AI features explain where users should double-check the output?

PART 5

Growth

Chapter 15. Partnerships and Ecosystems

“If you want to go fast, go alone. If you want to go far, go together.”

— proverb

Two partnerships

In the same year, Brindlepath signed two partnerships that looked equally promising on paper.

The first was with a large consulting firm that advised health systems on care transitions. The firm’s partners loved Brindlepath’s product and agreed to recommend it to their clients. A press release went out. A joint webinar drew a big audience. Twelve months later, the partnership had produced one qualified lead. The consultants were paid for advice, not software, and recommending Brindlepath did nothing for their own targets.

The second was with a regional home-health company that cared for many of the patients Brindlepath’s customers discharged. It began as a modest technical integration: when a navigator assigned a patient to home health, the referral flowed directly into the home-health company’s system, with the discharge summary attached, instead of by fax. Within a year, the home-health company was asking its own hospital partners to adopt Brindlepath, because the integration cut its intake time in half. Three new customers came through that channel.

The difference was not enthusiasm. Both partners were enthusiastic. The difference was that the second partnership created value for both companies, every day, in the normal course of their work. The first depended on goodwill.

Why partner at all

Partnerships extend what a product can do and where it can reach, without building everything yourself. They are worth pursuing when they do at least one of four things:

1. **Complete the product.** A partner fills a capability gap your customers need but that is not your core, such as Brindlepath's integration with a scheduling partner when it retired its own scheduling product in Chapter 7.
2. **Extend reach.** A partner gives you access to customers, channels or markets you could not reach efficiently alone.
3. **Strengthen the outcome.** A partner sits elsewhere in the customer's workflow, so connecting to them makes the end result better, as with the home-health integration.
4. **Accelerate innovation.** A partner brings technology, data or expertise that lets you build something new faster, such as an AI model provider or a research institution.

Partnerships that do none of these are usually press releases.

Types of partnerships

Partnerships come in several forms, each with different economics and demands.

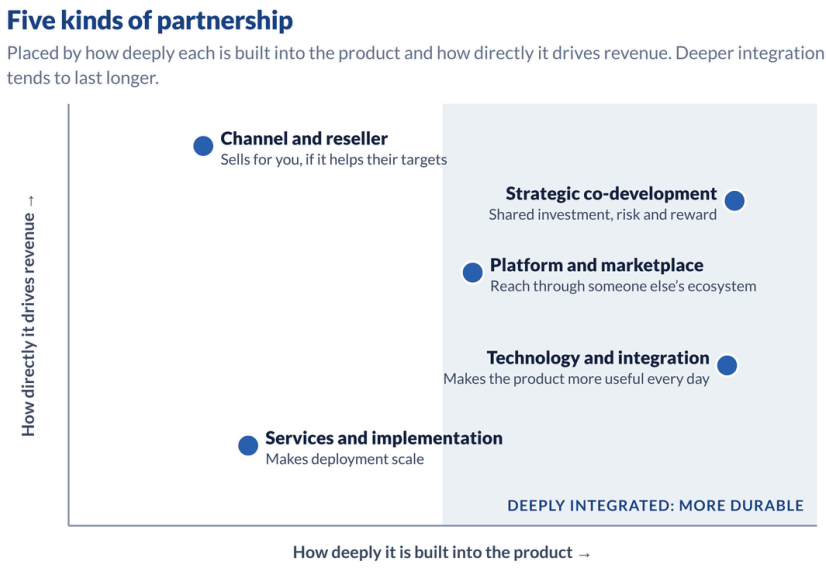


Figure 15.1. Types of partnership, by how deeply they are integrated into the product and how much revenue they are expected to drive. Deeper integration usually means stronger, more durable partnerships.

Technology and integration partners connect products so data and workflows flow between them: record systems, home-health and post-acute providers, identity providers, analytics platforms. They are the most common and often the most valuable, because they make the product more useful every day.

Platform and marketplace partners distribute your product through their ecosystem: an app marketplace attached to a record system, a cloud provider's marketplace, or a larger vendor's catalog. They can extend reach dramatically, at the cost of fees, rules and some dependence on the platform owner.

Channel and reseller partners sell your product, often bundled with their own services. They work best when selling your product helps them meet their own goals.

Services and implementation partners help customers deploy and get value from your product. For complex products, they can make implementation scalable.

Strategic and co-development partners build something new together, sharing investment, risk and rewards. They require the most trust and governance.

Choosing the right partners

Partnerships take real time and attention from small teams, so be selective. Before committing, assess each candidate on a few dimensions:

- **Mutual value:** what does each side gain, concretely and measurably? If you cannot write down the partner's gain in one sentence, the partnership will drift.
- **Customer overlap:** do you serve the same customers, so that working together helps them?
- **Workflow fit:** does the partnership connect naturally to how customers already work?
- **Incentive alignment:** does working with you help the partner's people hit their own targets?
- **Capability and commitment:** does the partner have the people, technology and executive sponsorship to follow through?
- **Risk:** data sharing, reputation, dependency, competition and regulatory exposure.

The consulting firm in the opening story scored well on customer overlap and poorly on incentive alignment. The home-health company scored well on almost everything, especially workflow fit.

Building the partnership

Once you choose a partner, treat the partnership like a product.

Agree on shared goals and measures. Referrals made, integrations live, patients transferred, time saved, joint revenue. Write them down and review them together.

Start small and prove value. Begin with a pilot: one integration, one region, one shared customer. Expand when it works.

Name owners on both sides. Every partnership needs a person on each side who is accountable for its success and has the authority to make decisions.

Make it easy to work together. Clear technical documentation, a sandbox for testing, simple commercial terms and a regular meeting rhythm.

Get the agreements right. In healthcare, any partnership that involves patient data needs the right data agreements, including business associate agreements where appropriate, as Chapter 11 described. Agree on data use, security, liability, intellectual property and how the partnership can end.

Building an ecosystem

As a product grows, its partnerships can become an **ecosystem**: a network of partners, customers and developers who build on and around the product. Ecosystems create value that no single company could: more integrations, more use cases, more reasons for customers to stay.

Ecosystems usually grow from a few foundations:

- **Open, well-documented interfaces** that partners can build on without lengthy negotiation, using industry standards where they exist, such as HL7 FHIR in healthcare.
- **A partner program** with clear tiers, benefits and requirements.
- **A marketplace or directory** where customers can find partner integrations.
- **Shared success stories** that show customers and prospective partners what is possible.

Brindlepath's ecosystem began with three integrations and grew to more than twenty in three years: home-health and post-acute providers, transportation services, community organizations that helped patients with food and housing, and analytics partners. Each integration made the core product more valuable to every customer.

Navigating partnership challenges

Even good partnerships hit trouble. The most common problems are predictable.

Misaligned incentives. Priorities shift and the partnership stops serving one side. Review mutual value regularly, not just at signing.

Competing priorities. The partnership is your top priority and the partner's tenth. Executive sponsorship on both sides helps; so does making the partnership part of each side's measured goals.

Integration decay. Partners change their systems and integrations break. Monitor every integration, with alerts when data stops flowing, and agree on how changes will be communicated.

Becoming a competitor. A partner may build what you offer, or you may build what they offer. Discuss the boundaries openly and early, and plan for the possibility.

Dependence on a platform. If a large share of your business flows through one platform or marketplace, its rule changes can hurt you badly. Diversify where you can.

Ending well. Some partnerships should end. Plan how customers will be supported, how data will be handled and how integrations will be retired, as Chapter 7 described for products.

Brindlepath in practice: from referral to ecosystem

The home-health integration described at the start of the chapter grew into Brindlepath's partner program over two years, and the way it grew is instructive.

It began with one integration, built to solve one shared problem: referrals sent by fax lost information and took days. The first version did one thing: when a navigator selected home health for a patient, the referral, with the discharge summary and medication list, appeared in the home-health company's intake system within minutes. Both companies agreed on two measures, the time from referral to first home visit and the share of referrals that needed a follow-up phone call for missing information, and reviewed them monthly.

The results were strong enough that the home-health company started recommending Brindlepath to the hospitals it worked with, and other post-acute providers began asking for the same integration. Rather than build each one from scratch, Brindlepath turned the work into a standard referral interface based on HL7 FHIR, published its documentation, and created a simple partner program with two tiers: integrated partners, who had built and tested the interface, and strategic partners, who also shared measures and planning with Brindlepath.

The program grew to more than twenty partners: home-health agencies, skilled nursing facilities, transportation services and community organizations that helped patients with food and housing. Each partner integration was monitored, with alerts when referrals stopped flowing, and each had a named owner at Brindlepath.

Two partnerships ended along the way. One partner was acquired by a company that built a competing product; Brindlepath gave customers six months' notice and helped them move referrals to other providers. Another never produced meaningful referral volume and was retired by mutual agreement. Ending partnerships well, as Chapter 15 recommends, kept the program's reputation intact.

Common mistakes

Partnering for the press release. Announcements are easy. Partnerships that change nothing in either company's daily work quietly fade.

Ignoring the partner's incentives. If working with you does not help your partner's people hit their targets, enthusiasm will not survive the next quarter.

Too many partners, too soon. Each partnership needs attention. A small company with twenty partners usually has twenty shallow ones.

No owner on either side. Partnerships without a named, accountable person on each side drift as soon as the first sponsor moves on.

Integrations without monitoring. A partner changes an interface, data stops flowing, and nobody notices until customers complain.

Becoming dependent on one platform. When a large share of revenue flows through one marketplace or platform, its rule changes become your crisis.

Try this

List your current and candidate partners. For each, write one sentence on what you gain and one on what they gain, then score them from one to five on customer overlap, workflow fit, incentive alignment and commitment. Any partnership where you cannot write the partner's sentence, or that scores low on incentive alignment, deserves a frank conversation or a graceful exit. For your strongest partnership, agree on two shared measures and a date to review them together.

How AI changes this

AI model providers are a new kind of critical partner. Many products now depend on third-party AI models. That partnership carries the same questions as any other, mutual value, risk and dependency, plus new ones: how the provider handles your customers' data, whether it trains on it, how model

changes will affect your product's quality, and what happens if prices change or a model is retired. Treat model providers as strategic partners, and design your product so you can switch providers if you must.

Integrations get easier, and more important. AI tools can help generate integration code and map data between systems, lowering the cost of new integrations. At the same time, AI features are only as good as the data they can reach, so integrations that bring in more context become more valuable.

AI agents will act across partner boundaries. As customers adopt AI agents that work across several systems, partnerships will increasingly be about how agents from different vendors can safely work together: what data they can exchange, what actions they can take, and who is accountable when something goes wrong. Emerging standards, such as protocols for connecting AI assistants to tools and data, are worth watching.

Partnership management benefits from AI too. Assistants can monitor partner integrations, summarize partner performance before reviews and spot co-selling opportunities across shared customers.

Chapter checklist

- [ ] Does each partnership complete the product, extend reach, strengthen the outcome or accelerate innovation?
- [ ] Can you state, in one sentence each, what both sides gain?
- [ ] Do partners' own incentives reward them for working with you?
- [ ] Do partnerships start with a small pilot and expand on evidence?
- [ ] Does each partnership have named owners on both sides, shared measures and a regular review?
- [ ] Are data agreements, including business associate agreements where needed, in place before data flows?
- [ ] Are partner integrations monitored, with alerts when data stops flowing?
- [ ] Have you discussed openly where each side might compete?

- [ ] Is your product designed so you could switch AI model providers if you had to?

Chapter 16. Scaling the Product and the Company

“Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations.”

— Melvin Conway

Twelve customers, then sixty

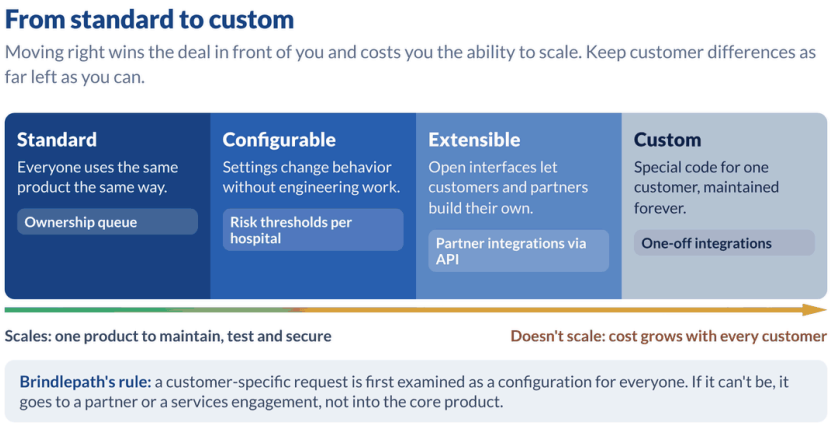
When Maya joined Brindlepath, it had twelve customers. Three years later, it had sixty. Almost nothing that worked at twelve worked at sixty without change.

At twelve customers, each implementation was a small craft project: an engineer spent weeks on each customer’s record integration, and the product had grown dozens of customer-specific settings, some of which nobody remembered the purpose of. At sixty, that approach would have needed a hundred engineers just for implementations. At twelve, Maya knew every customer’s chief nursing officer by name. At sixty, she could not. At twelve, two product teams coordinated over lunch. At sixty, seven teams needed a planning rhythm, shared standards and clear ownership, or they would trip over each other every week.

Scaling is not just doing more of the same. It is changing how the product and the company work, so that growth makes things better instead of breaking them.

Configure, don’t customize

The single most important product decision in scaling business software is how you handle the differences between customers. Every customer believes its workflow is unique. Some differences are real; many are habits. How you respond shapes your margins, your speed and your product’s quality.



> Figure 16.1. The spectrum from standard to custom. Moving right wins individual deals and loses the ability to scale. Push customer differences as far left as you can.

At one end, every customer uses the same product in the same way. At the other, every customer has its own code. Healthy products sit near the left: one product, shared by all, with **configuration**, settings that customers or your team can change without engineering work, for the differences that genuinely matter, and **extension points**, such as open interfaces, where customers or partners can build what is unique to them.

Customization, building special code for one customer, feels like good service. It wins the deal in front of you. It also creates code that must be maintained, tested, secured and upgraded for that one customer forever, which is why Chapter 6’s three smallest customers were losing money. Brindlepath adopted a simple rule: a request for something customer-specific is first examined as a potential configuration available to everyone; if it cannot be, it goes to a partner or a services engagement, not into the core product.

Scaling the technology

Product leaders do not need to design architectures, but they do need to understand the trade-offs, because scaling problems show up as product problems: slow screens, outages, failed integrations, delayed features.

Multi-tenancy. Most SaaS products serve many customers from shared infrastructure, with each customer's data kept strictly separate. It is far more efficient than running separate copies, and it requires careful design to make sure no customer can ever see another's data.

Standard integrations. Replace one-off integrations with a standard connector, configured per customer. Brindlepath's record connector, from Chapter 7, cut the time to integrate a new hospital from months to weeks.

Reliability targets. As customers depend on the product, reliability becomes a feature. Many teams set **service level objectives (SLOs)**, explicit targets for availability and speed, and track how much of their allowed unreliability, their **error budget**, they have used. When the budget runs low, the team shifts effort from features to reliability. This turns an argument into a rule.

Observability. At scale, you cannot rely on customers to tell you when something is wrong. Monitoring, logging and alerting should show problems before customers notice them, especially for integrations, where a silent failure can mean patients missing from the queue.

Performance and cost. As usage grows, so do infrastructure and AI costs. Track cost per customer, as Chapter 6 recommended, and treat sudden increases as bugs.

Balancing quality with growth

Growth puts pressure on quality from every direction: more customers, more features, more engineers, more code. Without deliberate effort, quality erodes slowly and then all at once.

A few practices protect it:

- **Protect the capacity allocation** from Chapter 5. Reliability, security and technical debt get their share every quarter, especially when growth is fastest.
- **Automate testing and release,** so quality checks scale with the team rather than depending on heroics.
- **Use feature flags and staged rollouts** to limit the impact of any single change.
- **Watch quality measures alongside growth measures:** incident counts, error budgets, support tickets per customer, time to resolve.
- **Do blameless reviews** after every significant incident, and fix the system, not just the symptom.

Compliance at scale

Compliance obligations grow with the business. New customers bring new security reviews. New states and countries bring new privacy laws. Larger customers ask for independent certifications. AI features bring new regulations.

Scale compliance the way you scale everything else: make it systematic.

- **Invest in certifications before you need them.** A SOC 2 report or HITRUST certification takes months. Start before the big deal that requires it.
- **Build compliance into the platform.** Audit logging, access controls, data retention and regional data storage built once, used everywhere.
- **Maintain the trust package** from Chapter 11 as a living product.
- **Track regulatory change.** Assign someone to watch the regulations that affect you and translate changes into product requirements early.

Expanding market reach

Growth eventually requires reaching beyond the first market. There are a few classic paths, each with its own risks.

Deeper in the same customers. More users, sites and modules within existing customers. Usually the cheapest growth, built on the customer success work in Chapter 14.

More customers in the same segment. Repeating what works with similar customers. Depends on a repeatable sales and implementation process.

Adjacent segments. New types of customers with similar problems. For Brindlepath, that meant large physician groups and health plans, which also needed to coordinate care after hospital discharges. Adjacent segments often need different buyers, messages and features, so treat each as a new product-market fit question, using the discovery methods in Chapters 2 and 3.

New geographies. New countries bring new regulations, languages, payment systems and buying habits. In healthcare, they also bring entirely different health systems. International expansion is rarely a matter of translation.

Expand one direction at a time, and only when the core is strong. A company that expands while its core product is still struggling usually ends up with two struggling products.

Scaling the organization

Melvin Conway's observation at the top of this chapter has become known as **Conway's law**: the structure of a system tends to mirror the communication structure of the organization that builds it. If three teams must coordinate to change one workflow, the product will show the seams.

The practical lesson is to design teams around the product you want. Brindlepath reorganized at about forty engineers from teams organized by technology layer to teams organized around customer journeys: the navigator's day, discharge and integration, outcomes and reporting, and the patient's experience, with a platform team providing shared services. Each journey team could deliver most changes on its own.

How product practices change as the company grows ILLUSTRATIVE

What works at one stage becomes a constraint at the next. Change practices just before they break.

	Early 1–2 teams, ~10 customers	Growing 3–6 teams, ~50 customers	Scaling 7+ teams, 100+ customers
Strategy	In the founders' heads	One page, shared widely	Strategy plus portfolio and horizons
Teams	Everyone does everything	Trios, teams by customer journey	Journey teams plus a platform team
Roadmap	A list, updated weekly	Now, Next, Later by outcome	Team roadmaps within one strategy
Customers	Leader knows each one	Advisory group, field visits	Research program, health scores
Customer differences	Custom work wins deals	Configuration over customization	Extension points and partners
Quality	Heroics	Automated tests, flags, reviews	Service level objectives, error budgets
Trust	Answer questionnaires as they come	A maintained trust package	Certifications, compliance in the platform
AI	Individuals try tools	Shared tools, evaluation for AI features	AI platform, governance, agent owners

Figure 16.2. How product practices change as the company grows (illustrative). What works at one stage becomes a constraint at the next. The skill is to change practices just before they break, not long after.

As the organization grows, the product leader's job changes too:

- **From doing to enabling.** Less time on individual features, more on strategy, hiring, coaching and the system that teams work within.
- **From conversations to written communication.** Strategy pages, decision logs and product briefs replace hallway agreements.
- **From one roadmap to many.** Each team owns its roadmap within a shared strategy and portfolio view.
- **From knowing every customer to knowing the patterns.** Customer advisory boards, research programs and data replace personal relationships with every account, though product leaders should still talk to customers every week.

Brindlepath in practice: the configuration rule

When Brindlepath adopted its configuration rule, the product had accumulated 47 customer-specific code paths, settings and integrations over its first years. Dev Raman's team estimated that maintaining them consumed close to a quarter of engineering capacity, mostly in testing and upgrades.

The team sorted them into three groups. Nineteen could become configuration available to every customer: for example, a customer-specific risk threshold became a setting any hospital could adjust, with sensible defaults. Fourteen belonged with partners or services: custom reports, for instance, moved to the outcomes module's export and a reporting partner. And fourteen could simply be retired, because the customers who had asked for them no longer used them, or had never used them at all.

The migration took most of a year, done customer by customer, with notice and support. Some conversations were difficult; one customer had built a workflow around a special integration and needed months of help to move. But by the end, the share of engineering capacity spent on customer-specific maintenance had fallen to a small fraction of its former level, and the time to onboard a new customer had dropped sharply.

The rule also changed sales conversations. Instead of promising custom work, sellers could show prospects the configuration options available to every customer, the partner ecosystem for specialized needs, and the product's open interfaces. Prospects who needed something truly unique were pointed to services or partners. A few deals were lost. Many more were won faster, and every customer benefited from a product that improved for all of them at once.

Common mistakes

Saying yes to custom work to win deals. Each custom request looks small. Together they become the maintenance burden that slows every release and erodes your margins.

Scaling a product that has not found fit. More sales and marketing on top of weak retention accelerates churn, not growth.

Letting reliability become an afterthought. As customers depend on you, outages cost more. Service level objectives and error budgets turn reliability into a managed commitment.

Expanding in every direction at once. New segments, new geographies and new products together stretch a growing company past breaking point.

Ignoring Conway's law. Teams organized by technology layer produce products whose seams show to customers.

Keeping early-stage habits too long. Practices that worked with two teams, such as decisions in the hallway and a roadmap in one person's head, break quietly at seven.

Try this

Make a list of every customer-specific setting, integration and code path in your product. For each, ask whether it could become a configuration available to every customer, whether it belongs with a partner, or whether it should be retired. Estimate the engineering time each one costs per year. Then pick the three most expensive and put a plan for each on the roadmap. Most teams find that a handful of special cases consume a surprising share of their capacity.

How AI changes this

AI changes the economics of scale. AI tools can automate parts of implementation, support, testing and documentation, reducing the people needed per customer. Companies can serve more customers with smaller teams than ever before. That is an opportunity to grow efficiently, but only if the core product is standardized enough for automation to work. AI does not rescue a product built on customization.

AI costs scale with usage. Unlike most software costs, AI inference costs grow with every request. At scale, small inefficiencies become large bills. Engineering for AI cost, by using smaller models where they are good enough, caching results and limiting unnecessary calls, becomes a real discipline.

AI governance must scale too. When many teams build AI features, they need shared standards: approved models and providers, evaluation requirements, safety reviews and monitoring. Chapter 17 describes the practices; scaling means making them part of the platform rather than leaving each team to invent its own.

AI can help leaders see the whole system. Summaries of incidents, customer feedback, delivery measures and roadmap progress across many teams help a product leader spot patterns that would otherwise take weeks to assemble. The judgment about what to do remains human.

Chapter checklist

- [ ] Do you handle customer differences through configuration and extension points rather than custom code?
- [ ] Is there a rule for customer-specific requests: configuration for all, partner or services, but not core customization?
- [ ] Do you have service level objectives and an error budget that shifts effort to reliability when needed?
- [ ] Will monitoring tell you about problems, especially silent integration failures, before customers do?
- [ ] Is cost per customer, including AI cost, tracked and acted on?
- [ ] Are certifications started before the deals that will require them?
- [ ] Are you expanding in one direction at a time, from a strong core?
- [ ] Are teams organized around customer journeys, with a platform team for shared services?
- [ ] Are AI standards, such as approved providers, evaluation and monitoring, built into the platform?

Chapter 17. Building AI Products and Leading with AI Agents

“AI is the new electricity.”

— Andrew Ng

The demo that was almost perfect

Brindlepath’s Horizon 3 team, introduced in Chapter 7, had a bold idea: an AI agent that would read a high-risk patient’s discharge summary, medication list and recent notes, then draft a post-discharge plan for the navigator, covering who to call, what to ask, which follow-up appointments were needed and which warning signs to watch for. If it worked, a navigator could prepare for a call in two minutes instead of fifteen.

The first demo, to the leadership team, was stunning. The agent produced clear, well-organized plans in seconds. The CEO wanted to announce it at the industry’s biggest conference, six weeks away.

Maya asked for one thing first: an evaluation. The team assembled 300 real, anonymized discharge summaries and asked two experienced nurses to review the agent’s plans against the source documents. The plans were good, most of the time. But roughly one plan in thirty contained something that was not in the record: a medication the patient was not taking, or a follow-up appointment that had never been ordered. Every one of those errors was written as confidently as everything else.

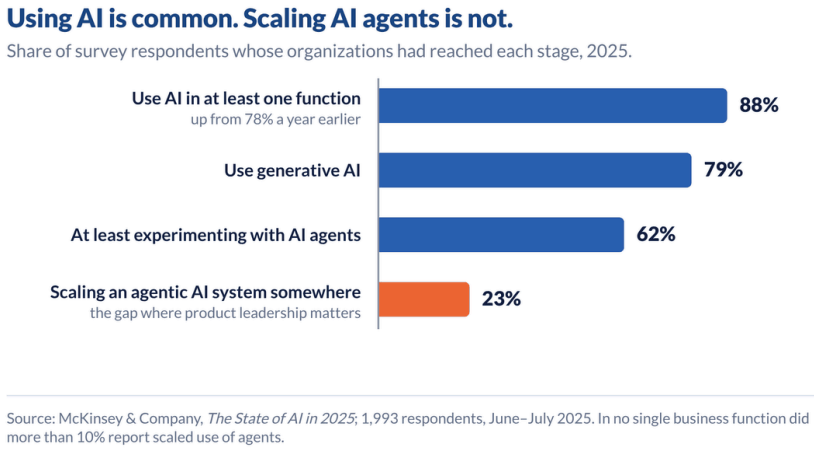
In a demo, one error in thirty is invisible. In a product used for thousands of patients a month, it is dozens of dangerous mistakes every week.

The team did not abandon the idea. They changed it. The agent now cited the source line for every statement, flagged anything it could not find in the record, and presented its plan as a draft the navigator reviewed and confirmed, never as an instruction. They built an evaluation set that ran before every change, and monitored errors in production. They launched four months later, with one customer, and expanded as the evidence came in.

This chapter is about building AI products that survive contact with reality, and about leading teams in which AI agents do real work.

Where organizations are

AI adoption has moved fast. In McKinsey's 2025 global survey, 88 percent of respondents said their organizations used AI in at least one business function, and 62 percent said they were at least experimenting with AI agents. Far fewer had scaled them: 23 percent reported scaling an agentic AI system somewhere in their enterprise, and in no single business function did more than 10 percent report scaled use of agents.[1]



Source: McKinsey & Company, *The State of AI in 2025*; 1,993 respondents, June–July 2025. In no single business function did more than 10% report scaled use of agents.

Figure 17.1. From using AI to scaling AI agents. Most organizations use AI somewhere; few have scaled agents. Source: McKinsey & Company, The State of AI in 2025.

The gap between experimenting and scaling is where product leadership matters most. The technology to build an impressive demo is widely available. The discipline to build something reliable, safe, affordable and genuinely useful is not.

When AI is the right tool

AI is not the answer to every problem, and using it where it does not fit creates cost and risk for no benefit. Before building an AI feature, ask what kind of problem it is.

AI tends to fit well when:

- the work involves understanding or producing language, images or other unstructured content;
- inputs vary widely and cannot be handled by simple rules;
- a good draft saves significant time, even if a person reviews it;
- some errors are acceptable, or can be caught by review;
- there is enough data, or enough examples, to evaluate quality.

AI tends to fit poorly when:

- the answer must be exactly right every time and can be computed with rules;
- the stakes of an undetected error are severe and there is no practical review step;
- the task is simple enough that ordinary software does it cheaply and reliably;
- you cannot measure whether the output is good.

Brindlepath's discharge-plan agent was a good fit for drafting, because summaries were long and varied and a draft saved real time, and a poor fit for acting alone, because an undetected error could harm a patient. The product design followed from that distinction.

How much should AI do on its own?

The most important design decision for an AI feature is its level of **autonomy**: how much it does without a person's involvement. A useful way to think about it is as a ladder.

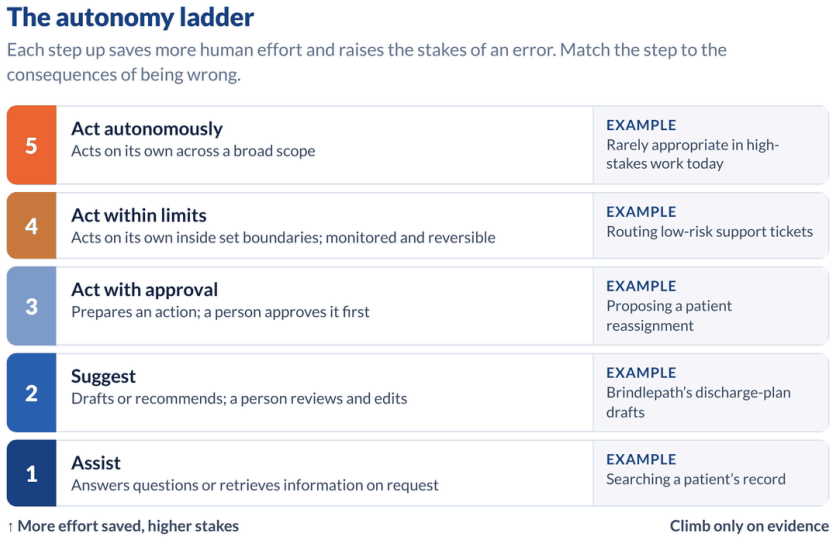


Figure 17.2. The autonomy ladder. Each step up saves more human effort and raises the stakes of an error. Match the step to the consequences of being wrong, and climb only on evidence.

1. **Assist.** AI answers questions or retrieves information when a person asks. The person decides what to do with it.
2. **Suggest.** AI proposes a draft or recommendation; a person reviews, edits and accepts or rejects it.
3. **Act with approval.** AI prepares an action, such as sending a message or updating a record, and a person approves it before it happens.
4. **Act within limits.** AI acts on its own within defined boundaries, with monitoring and the ability for people to review and reverse.
5. **Act autonomously.** AI acts on its own across a broad scope. Rarely appropriate in high-stakes domains today.

Match the step to the consequences of being wrong. Drafting a patient-education handout might reasonably sit at *suggest*. Updating the ownership of a patient in the queue might sit at *act with approval*. Changing a medication should not sit anywhere on this ladder without a clinician's decision. Start lower than you think you need to, measure, and climb only when the evidence supports it.

Designing AI experiences people can trust

AI features behave differently from traditional software. They are probabilistic: the same input can produce different outputs, and the output is sometimes wrong in ways that look right. Good AI design makes those properties visible and manageable.

Show the sources. Let people see where an answer came from. Brindlepath's agent cites the line in the discharge summary behind every statement, so a navigator can verify it with one click.

Signal uncertainty. Flag what the AI could not find or is unsure about, instead of letting it guess confidently.

Make correction easy. Editing a draft should be faster than writing from scratch. If it is not, people will stop using the feature.

Keep people in control. Make it clear what the AI has done and what it will do next, and let people stop, undo or override it.

Label AI output. People should always know when content was produced by AI.

Capture feedback. Every acceptance, edit and rejection is a signal about quality. Collect it, with appropriate privacy protections, and use it to improve.

Design for the reviewer's time. As Chapter 12 noted, AI that saves the author's time but costs the reviewer more is not a gain. Measure total time, not just generation time.

Evaluation: the discipline that separates demos from products

Traditional software is tested with expected outputs: given this input, the result must be exactly that. AI features need **evaluation**: systematic measurement of output quality across many realistic examples, against criteria people agree on.

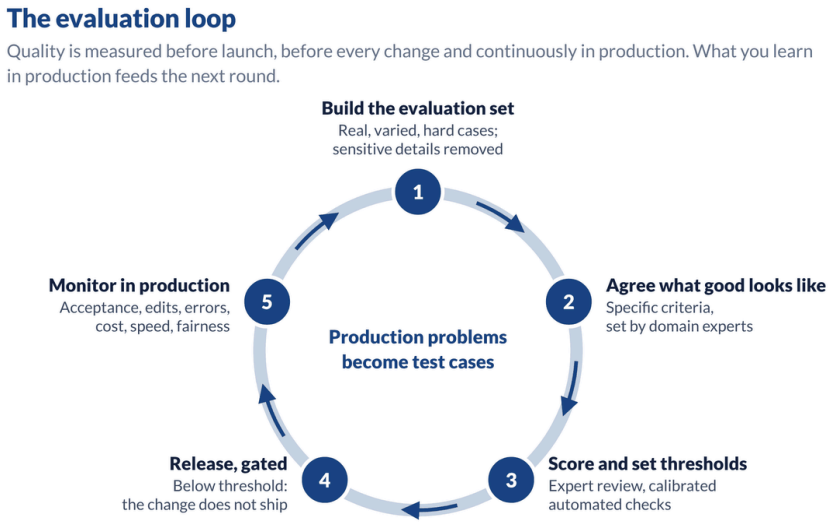


Figure 17.3. The evaluation loop. Quality is measured before launch, before every change and continuously in production, and what you learn in production feeds the next evaluation set.

A practical evaluation program has five parts.

1. An evaluation set. A collection of realistic inputs, including hard and unusual cases, ideally drawn from real data with sensitive details removed. Brindlepath's started with 300 discharge summaries and grew to over 1,000, deliberately including the messiest records the team could find.

2. Clear criteria. What does good look like? For the discharge plan: every statement supported by the record; no missing high-risk medications; correct follow-up appointments; plain language. Criteria should be specific enough

that two reviewers agree.

3. Scoring. Expert human review is the gold standard, especially in clinical settings. Automated checks, and AI models used as graders under careful supervision, can scale evaluation, but should be calibrated regularly against human judgment.

4. Thresholds. Decide in advance what quality is good enough to launch, and what change in quality blocks a release. For Brindlepath, any unsupported clinical statement in the evaluation set blocked a release until it was understood.

5. Production monitoring. Track quality in real use: acceptance and edit rates, user-reported errors, sampled expert review, cost and speed. Real-world inputs drift over time, and model updates can change behavior. Feed problems you find in production back into the evaluation set.

Evaluation is not a one-time gate. It is the core quality process for any AI product, and it deserves the same investment as testing does for traditional software. A team that cannot measure its AI feature's quality cannot improve it, and should not ship it.

Grounding AI in your data

General-purpose models know a great deal about the world and nothing about your customer. Most useful AI features combine a model with the right context: the customer's records, documents, policies and history. A common approach, often called **retrieval-augmented generation**, finds the relevant information and provides it to the model with each request, instructing it to answer from that information.

Grounding has several benefits: answers reflect the customer's actual situation, sources can be cited, and errors are easier to detect. It also means your data quality, integrations and permissions become part of your AI quality. An agent that can see the wrong patient's record, or none of the right ones, will produce the wrong answer however good the model is. The integration work in Chapters 15 and 16 is AI work too.

Cost, speed and model choice

Every AI request has a cost and a delay. At small scale, they are easy to ignore. At scale, they shape the product and its margins, as Chapter 6 warned.

- **Use the smallest model that meets the quality bar.** Larger models are often more capable and more expensive. Many tasks are handled well by smaller, faster, cheaper models.
- **Don't call a model when you don't need to.** Cache results, avoid regenerating unchanged content, and use ordinary code for anything deterministic.
- **Design for latency.** Show progress, stream results, or run work in the background so people are not left waiting.
- **Keep the option to switch.** Model providers change prices, capabilities and terms. An architecture that can swap models, with an evaluation set to confirm quality after a switch, protects you, as Chapter 15 recommended.

Safety, security and responsibility

AI features introduce risks that product leaders must own, building on Chapter 11.

Harmful errors. In domains like healthcare, wrong outputs can hurt people. Keep humans in the loop for consequential decisions, evaluate rigorously, and monitor in production.

Bias and fairness. Models can perform worse for some groups of people than others. Evaluate quality across the groups your product serves, including by age, language and other relevant characteristics, and address gaps before launch.

Manipulation and misuse. AI systems can be tricked through their inputs, for example by instructions hidden in a document an agent reads. Limit what agents can access and do, treat external content as untrusted, and require approval for consequential actions.

Privacy. Know exactly what data reaches each model, where it is processed, whether it is retained, and whether it is used for training. Get the agreements in place before data flows.

Regulation. Requirements are evolving: medical device rules for clinical AI, the EU AI Act's obligations for high-risk systems, state laws and sector guidance. The NIST AI Risk Management Framework offers a practical structure for governing, mapping, measuring and managing AI risks.[2] Involve legal and compliance partners early, as Chapter 11 recommended.

Leading teams that include AI agents

AI agents increasingly do real work inside product organizations: drafting research syntheses, triaging support tickets, monitoring dashboards, writing and reviewing code, preparing release notes. Treating them well is now part of leadership.

A practical way to think about it: manage an agent the way you would manage a new team member with specific, limited responsibilities.

- **Give it a job description.** What exactly does it do, for whom, with what inputs, and what does good output look like?
- **Give it an owner.** A named person is accountable for each agent's instructions, permissions, quality and failures, as Chapter 8 recommended.
- **Limit its permissions.** Grant access only to the data and actions its job requires. Review those permissions regularly.
- **Review its work.** Sample its output, track its quality, and correct it. Agents do not improve by themselves; their owners improve them.
- **Make it visible.** Everyone should know which work is done by agents, and how to raise a concern about it.
- **Keep humans accountable for decisions.** Agents can gather, draft, recommend and, within limits, act. Accountability for outcomes, especially those affecting customers, money, safety or people, stays with people.

The product leader's job is to design the system in which people and agents work together well: deciding which work suits agents, where human judgment must stay, and how quality is assured across both. That is the "orchestrator" role from Chapter 1, in practice.

From idea to AI product

Pulling it together, a practical path for building an AI product or feature looks like this:

1. **Start with the problem,** using the discovery methods in Chapters 2 and 3, not with the technology.
2. **Check the fit:** is AI the right tool, and what is the cost of an error?
3. **Choose the autonomy level** based on the consequences of being wrong.
4. **Build the evaluation set early,** before the feature, with experts defining good.
5. **Prototype quickly** and measure against the evaluation set, not against a demo.
6. **Design the experience for trust:** sources, uncertainty, easy correction, control.
7. **Pilot with a small group,** monitor closely, and expand on evidence.
8. **Monitor quality, cost and safety in production,** and feed what you learn back into evaluation.

Brindlepath's discharge-plan drafts followed that path. In the story, navigators accepted about seven in ten plans with minor or no edits, preparation time per high-risk patient fell substantially, and the share of high-risk patients contacted within 24 hours rose at the pilot customer. Those results are part of the fictional example; your numbers will depend on your product and data. The method is what transfers.

Brindlepath in practice: the evaluation set

The evaluation set for the discharge-plan agent became one of the most valuable assets the company owned.

It began with 300 discharge summaries from two pilot customers, used with permission under the customers' agreements, with names, dates and other identifying details removed. The team deliberately over-sampled hard cases: patients with long medication lists, summaries with contradictory information, handwritten notes scanned into the record, and patients transferred between hospitals.

Two experienced nurses wrote the quality criteria with the team. Every statement in a plan must be supported by the record. Every high-risk medication must be mentioned. Follow-up appointments must match what was ordered. Warning signs must be appropriate to the diagnosis. The language must be plain enough for a new navigator to act on. Each criterion was scored pass or fail, so that two reviewers would agree.

The nurses reviewed the first runs by hand. As the set grew, the team added automated checks, for example matching every medication in a plan against the medication list, and an AI-assisted grader for criteria that were harder to automate. The grader's scores were compared each month against a sample reviewed by the nurses, and recalibrated when they drifted apart.

Every proposed change to the agent, whether a new instruction, a new model version or a new data source, ran against the full set before release. Twice, a change that looked better in a demo made the evaluation results worse, and was not released. Once, a model provider's update changed behavior subtly; the evaluation caught it before any customer saw it.

Production monitoring fed the set. Every plan a navigator flagged as wrong was reviewed, and the most instructive cases, with details removed, were added to the evaluation set. After a year it held over 1,000 cases and reflected the real variety of the work far better than any test the team could have invented.

Common mistakes

Starting with the model instead of the problem. "We should use AI" is not a product idea. A validated customer problem is where every AI feature should begin.

Judging quality by the demo. Demos show the best case. Evaluation across hundreds of realistic examples shows what customers will actually experience.

Too much autonomy, too soon. Letting AI act on its own before the evidence supports it turns occasional errors into customer incidents.

Hiding the AI. Unlabeled AI output, without sources, makes errors harder to catch and erodes trust when they are found.

Ignoring cost until the invoice arrives. AI costs scale with use. A feature that is affordable in a pilot can be ruinous at scale.

Agents without owners. An AI agent with broad permissions and nobody accountable for it is an incident waiting to happen.

Try this

Pick one AI feature you have built, or are considering. Write its evaluation plan using the template in the Appendix: purpose, autonomy level, cost of an error, evaluation set, criteria, thresholds, fairness checks, monitoring and owner. If you cannot fill in the evaluation set and the criteria, stop and build them before you write another line of the feature. Then write a job description, from the Appendix template, for one AI agent your team uses, and check that its permissions match it.

Chapter checklist

- [ ] Is each AI feature solving a real, validated problem, not a technology looking for one?
- [ ] Have you assessed whether AI is the right tool, and what an undetected error would cost?

- [ ] Is the autonomy level matched to the consequences of being wrong, and raised only on evidence?
- [ ] Do AI outputs show sources, flag uncertainty, invite easy correction and carry a clear label?
- [ ] Is there an evaluation set, with expert-defined criteria and launch thresholds, built before the feature?
- [ ] Does evaluation run before every change, and is quality monitored in production?
- [ ] Is the AI grounded in the right data, with permissions that match the user's?
- [ ] Are model cost and speed tracked, and could you switch providers if needed?
- [ ] Have you evaluated fairness across the groups you serve, and protected against manipulation?
- [ ] Does every AI agent have a job description, an owner, limited permissions and regular review?
- [ ] Are humans accountable for every decision that affects customers, money, safety or people?

---

1. McKinsey & Company, *The State of AI in 2025: Agents, Innovation, and Transformation*, survey of 1,993 participants, June–July 2025.
2. National Institute of Standards and Technology, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, January 2023.

CHAPTER 18

The Integrated Playbook

“Plans are worthless, but planning is everything.”

— Dwight D. Eisenhower

Three years later

Three years after her first Monday on the train, Maya Smith sat in the same quarterly business review with Ambermoor Health that had been on her overnight digest that first morning. The meeting was different now.

Ambermoor’s chief nursing officer opened with her own numbers: the share of high-risk patients with a confirmed owner within 24 hours had held above 85 percent for a year across seven hospitals. Missed first follow-ups had fallen. The finance team had used Brindlepath’s outcome report to show the board what avoided readmissions were worth. The navigators, she said, no longer spent their mornings on the phone trying to find out who owned whom.

Then she asked about the future. Could the discharge-plan drafts cover more patients? Could specialist offices see the handoff too? Could Brindlepath help with the health plan’s members as well as the hospital’s patients?

None of those questions was answered by a single feature, a single framework or a single chapter of this book. Each one touched the problem worth solving, the strategy and positioning, the numbers, the team, trust, customers, partnerships and AI. That is the nature of product leadership: the pieces only work when they work together.

This last chapter pulls the playbook into one picture, offers a practical rhythm for running it, and closes with some thoughts about the job.

One system, not seventeen chapters

Each chapter of this book covers one part of the work. In practice, they form a single system, where decisions in one area shape every other.

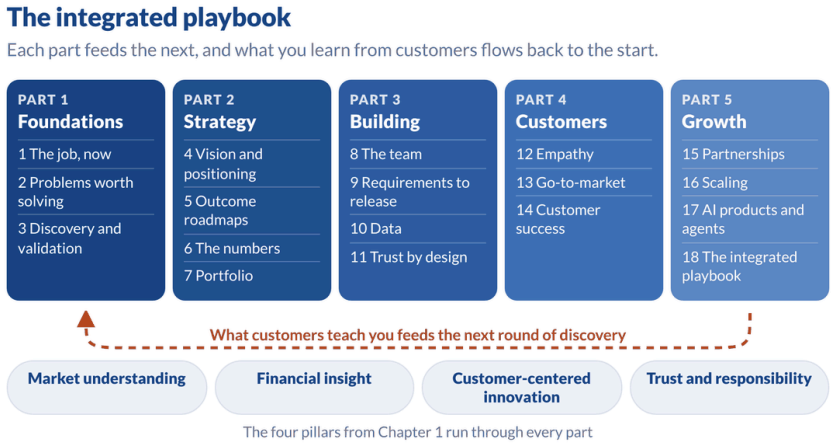


Figure 18.1. The integrated playbook. Each part of the book feeds the next, and what you learn from customers flows back to the start. The four pillars from Chapter 1 run through all of it.

Some of the connections Brindlepath lived through:

- **The problem shapes the business.** Choosing post-discharge ownership over scheduling (Chapter 2) made a value-based price possible (Chapter 6), because the value was measurable in avoided readmissions.
- **Strategy shapes the team.** The not-doing list (Chapter 4) freed a third of engineering capacity, which funded the reliability work (Chapter 5) and the integration platform (Chapter 7).
- **Economics shape the product.** Discovering that custom integrations made small customers unprofitable (Chapter 6) led to the standard connector (Chapter 16), which made partnerships possible (Chapter 15).

- **Trust opens markets.** Treating security and accessibility as product requirements (Chapter 11) shortened every sales cycle that followed (Chapter 13).
- **Customers close the loop.** A customer success manager's early warning (Chapter 14) revealed a data problem that became a product fix, and a renewal became an expansion.
- **AI raises the stakes on everything.** The discharge-plan agent (Chapter 17) depended on the integrations, the evaluation discipline, the trust practices and the team habits built over the previous three years.

When you face a decision, ask which other parts of the system it touches. Most product mistakes come from optimizing one part while ignoring the rest.

The four pillars, revisited

Chapter 1 described four pillars of product leadership. Looking back, each runs through the whole playbook.

Market understanding is not a phase. It begins with problem discovery and continues through every customer conversation, field visit, cohort analysis and renewal.

Financial insight is not a budget exercise. It informs which problems to solve, how to price, what to build, how to scale and which partnerships to pursue.

Customer-centered innovation is not a design department. It is the habit of every team member who watches users work and measures whether the product actually helped.

Trust and responsibility is not a compliance review. It is built into every requirement, every release, every AI feature and every partnership.

A product leader strong in all four, and able to connect them, is rare and valuable. Most of us are stronger in some than others. Know your weakest pillar and work on it deliberately, or hire people who are strong where you are not.

An operating rhythm

Frameworks are only useful if they are used regularly. The following rhythm, adapted from what worked at Brindlepath and in teams I have led, puts the playbook into practice.



> Figure 18.2. An operating rhythm for product leadership. Each cadence answers different questions, with different people in the room.

Weekly. Customer conversations for every product trio. Review of input metrics and health scores. Team planning and decision log updates. Thirty minutes of customer feedback review with customer success.

Monthly. Product review of outcomes against the roadmap. Review of AI feature quality and cost. Partner integration health check. A field visit for at least one trio.

Quarterly. Roadmap and capacity allocation review. Portfolio review with stage-appropriate measures. Business reviews with key customers. Launch reviews for anything released in the quarter. Retrospective on how the team is working.

Yearly. Strategy refresh: diagnosis, guiding policy, actions and the not-doing list. Vision check. Pricing and packaging review. Horizon allocation for the year ahead.

The rhythm matters more than the exact meetings. Without it, urgent work crowds out important work, and strategy becomes something you write once a year and forget.

Your first 90 days as a product leader

Many readers will use this book when starting a new product role. A practical plan for the first 90 days, drawn from Maya's experience and my own:

Days 1 to 30: listen and learn.

- Meet every member of the product, design and engineering teams one to one.
- Talk to at least ten customers, including some who left or said no.
- Spend time where the product is used, as Lena did in Chapter 12.
- Learn the numbers: revenue, margin, retention, cost to serve, as in Chapter 6.
- Read the existing strategy, roadmap and backlog, and the last year of decisions, if anyone wrote them down.
- Change very little.

Days 31 to 60: diagnose and align.

- Write your diagnosis: what is working, what is not, and the hardest problem the product faces.
- Test it with the leadership team, the product team and a few trusted customers.
- Identify one or two quick wins that demonstrate progress and build trust.
- Clean up the backlog, as Maya did in Chapter 5.

Days 61 to 90: decide and communicate.

- Write the one-page strategy, including the not-doing list.
- Agree on the North Star and input metrics.
- Rebuild the roadmap around outcomes.

- Set the operating rhythm.
- Tell the story, repeatedly.

Common mistakes

Optimizing one part of the system. A brilliant feature that the economics cannot support, or a strong strategy that the team cannot deliver, fails just as surely as a weak one.

Collecting frameworks. Frameworks help only when they are used regularly, by the whole team, to make real decisions. A rhythm matters more than a library.

Changing everything in the first month. New leaders who reorganize, rewrite the strategy and rebuild the roadmap before they understand the product lose the trust they need later.

Forgetting to look at the results. Plans are useful; results are what count. Build the habit of checking whether what you did made the difference you expected.

The job, honestly

Product leadership is a strange job. You are accountable for outcomes you cannot directly control. You have authority over very little and influence over almost everything. You are expected to be right about the future while knowing that much of what you believe is wrong. You spend your days choosing what not to do, and disappointing someone with almost every choice.

It is also one of the best jobs in business. You get to understand people's problems deeply, assemble talented teams to solve them, and see the results in the lives of the people you serve. In healthcare, that can mean a patient who gets the call she needed, from a navigator who knew she was his, because a team made a hundred careful decisions about how that should work.

AI will keep changing how the work is done. Agents will take on more of the gathering, drafting, analyzing and monitoring. The tools in this book will evolve. But the core of the job, understanding people, choosing well, building trust and owning the outcome, will remain human for a long time. The best product leaders of the next decade will be the ones who use every tool available to do that core job better, and never confuse the tools with the job.

I hope this playbook helps you do it.

Final checklist

- [ ] Do you consider how each major decision affects the rest of the system: problem, strategy, numbers, team, trust, customers, partners and AI?
- [ ] Do you know your weakest pillar, and what you are doing about it?
- [ ] Do you have a weekly, monthly, quarterly and yearly rhythm, and do you keep it?
- [ ] Does every product trio talk with customers every week?
- [ ] Is your strategy on one page, with a not-doing list, refreshed at least yearly?
- [ ] Do you look at the results, and change course when they tell you to?

Appendix: Templates

These templates are starting points. Copy them, adapt them to your product and your organization, and keep them short. A template that nobody fills in is worse than none.

A. Problem statement

[Who] need to **[goal or job]**, but **[obstacle]**, which leads to **[consequence, ideally measurable]**. Today they **[current workaround]**, which falls short because **[gap]**.

Check it:

- [ ] It names a specific person, not "users."
- [ ] It contains no solution.
- [ ] The consequence is something the customer cares about, in their terms.
- [ ] Evidence could prove it wrong.

See Chapter 2.

B. One-page product strategy

Vision. One sentence describing the world you want to create for customers.

Diagnosis. What is really going on in the market, and the hardest problem you face. Three to five sentences.

Where we play. The customers, problems and channels you will focus on.

How we win. The advantage that will make those customers choose you, and why competitors with the same AI tools cannot copy it in a quarter.

Guiding policy. Your overall approach, in one or two sentences.

Key actions. Three to five coherent actions for the next year.

Outcomes. The North Star and the input metrics you will use to know it is working.

Not doing. What you will stop or decline, with a sentence on why.

See Chapter 4.

C. Product brief

Title and owner. The name of the work and the product trio accountable for it.

Problem. What problem, for whom, with what evidence. Link the problem statement.

Outcome measure. The measure you expect to move, today's baseline, and the target.

Who it is for. Users, buyers and beneficiaries.

What we know. Evidence from discovery.

What we assume. The riskiest assumptions, and how and when you will test them.

Scope. What is in the first slice.

Out of scope. What is explicitly not included, and why.

Must include. Non-negotiable requirements: privacy, security, audit, accessibility, performance.

Open questions. What the team will answer as it builds.

Decision log. Dated decisions, with the reasoning.

See Chapter 9.

D. One-page roadmap

Now (committed)	Next (likely)	Later (exploring)
Problem or outcome, with its measure	Problem, with its measure	Problem only
…	…	…
Dated commitments, marked clearly		

Capacity allocation this quarter: new outcomes **%, reliability and performance** %, security and compliance **%, technical debt** %, discovery and experiments __%.

See Chapter 5.

E. Business case

1. **Problem and customers:** from discovery, with evidence.
2. **Revenue effect:** new customers, expansion, churn prevented, price supported. Show a low, likely and high case, with assumptions.
3. **Cost to build:** product, design and engineering time; third-party costs.
4. **Cost to run:** hosting, support, maintenance, AI model costs, partner fees, per customer and in total.
5. **Effect on unit economics:** gross margin, payback period, retention.
6. **Risks:** what has to be true, and what could go wrong.
7. **How we will know:** the measure, the threshold and the date you will check.

See Chapter 6.

F. Pilot agreement

- **Problem and measure:** the outcome being tested and how it will be measured.
- **Baseline:** the measure today.
- **Success threshold:** the result that justifies a purchase.
- **Scope and duration:** teams, sites, users and dates. Usually 60 to 90 days.
- **Responsibilities:** what each side provides, including data, people and time.
- **Price:** the pilot fee.
- **Decision-maker and date:** who decides, and when.
- **Expansion terms:** the price, scope and timeline that apply if the threshold is met.

See Chapters 3 and 13.

G. Launch readiness checklist

Product

- [ ] Acceptance criteria met; known issues documented.
- [ ] Outcome measures instrumented and baseline recorded.
- [ ] Feature flags and rollout plan in place.

Quality, security and trust

- [ ] Testing complete, including accessibility.
- [ ] Security review and, where relevant, privacy impact assessment done.
- [ ] For AI features: evaluation results meet thresholds; monitoring in place.

Customers

- [ ] Release notes and help content ready.
- [ ] Training materials ready; customer approvals obtained where required.

Go-to-market

- [ ] Launch tier agreed.
- [ ] Messaging by audience approved, with claims supported by evidence.
- [ ] Sales and customer success enabled: story, demo, objections, boundaries, practice.

Support and operations

- [ ] Support team trained; escalation paths clear.
- [ ] Rollback plan tested.

After launch

- [ ] Launch review scheduled, with the measures to check at 30, 60 and 90 days.

See Chapters 9 and 13.

H. Customer health score

Signal	Example measures	Weight
Outcomes	Customer's own outcome measure, trended	Highest
Adoption	Active users by site; usage trend	High
Relationship	Executive sponsor engaged; champion in place	Medium
Support	Open issues; ticket trend	Medium
Commercial	Payment timeliness; contract questions	Low

Every at-risk account has an owner and a written plan. See Chapter 14.

I. AI feature evaluation plan

- **Feature and purpose:** what the AI does, for whom.
- **Autonomy level:** assist, suggest, act with approval, act within limits, or act autonomously, and why.

- **Cost of an error:** what happens if the output is wrong and nobody notices.
- **Evaluation set:** source, size, and how hard and unusual cases are included; how sensitive data is removed.
- **Quality criteria:** specific, agreed by experts.
- **Scoring method:** expert review, automated checks, AI-assisted grading calibrated against people.
- **Launch thresholds:** what quality is required to launch; what blocks a release.
- **Fairness checks:** groups compared, and acceptable differences.
- **Production monitoring:** acceptance and edit rates, error reports, sampled review, cost and latency.
- **Owner:** the person accountable for quality.

See Chapter 17.

J. AI agent job description

- **Name and purpose:** what the agent does, in one sentence.
- **Owner:** the person accountable for it.
- **Inputs:** the data and sources it may use.
- **Outputs:** what it produces, and for whom.
- **Permissions:** the systems and actions it may access, and nothing more.
- **Human checkpoints:** which outputs or actions require review or approval.
- **Quality measures:** how its work is checked, and how often.
- **Escalation:** what it does when it is unsure, and how people raise concerns about it.
- **Review date:** when its job description and permissions will next be reviewed.

See Chapters 8 and 17.

K. Glossary

Acceptance criteria. The specific, testable conditions a user story must meet to be considered done.

Annual recurring revenue (ARR). The yearly value of active subscriptions, excluding one-time fees.

Autonomy level. How much an AI feature or agent does without a person's involvement: assist, suggest, act with approval, act within limits or act autonomously.

Business associate agreement (BAA). Under HIPAA, the contract required when a vendor handles protected health information on behalf of a healthcare provider or health plan.

Buying group. Everyone involved in a business purchase decision: users, champions, economic buyers, technical gatekeepers and influencers.

Cohort. A group of customers or users who started in the same period, tracked together over time.

Configuration. A setting that changes how the product behaves for a customer without new code.

Customer acquisition cost (CAC). Sales and marketing spending in a period, divided by the number of new customers won.

Discovery. The continuous work of learning what to build: understanding problems and testing solutions before and during delivery.

Error budget. The amount of unreliability a service is allowed under its service level objective. When it runs low, effort shifts to reliability.

Evaluation set. A collection of realistic examples, with agreed quality criteria, used to measure an AI feature's output before release and after every change.

Feature flag. A switch that turns a capability on or off for specific customers or users without a new release.

Grounding. Giving an AI model the specific, relevant information it should use to answer, such as a customer's records, rather than relying on its general knowledge.

Gross margin. Revenue minus the cost of delivering the service, as a percentage of revenue.

Gross revenue retention. The share of last year's recurring revenue kept from existing customers, ignoring upsells.

Input metric. A measure a team can move directly, which drives the North Star.

Jobs to be done. The progress a customer is trying to make in a particular circumstance, which they "hire" a product to help with.

Lifetime value (LTV). The gross profit a customer is expected to generate over the whole relationship.

Minimum viable product (MVP). The smallest thing that delivers real value to real users and tests the core assumption.

Net revenue retention (NRR). The share of last year's recurring revenue kept from existing customers, including expansion. Above 100 percent means existing customers are growing.

North Star metric. The single measure that best captures the value a product delivers to customers and leads long-term business results.

Outcome. A measurable change in customer behavior or results, as distinct from output, the features a team ships.

Payback period. How long it takes for a customer's gross profit to repay the cost of acquiring them.

Positioning. How you want the market to understand what your product is, who it is for and why it is better than the alternatives.

Pricing metric. The unit a customer pays for, such as a seat, a transaction or a patient covered.

Product trio. The product manager, product designer and engineering lead who share responsibility for discovery and the outcome.

Prompt injection. An attack in which instructions hidden in content an AI system reads attempt to change its behavior.

Psychological safety. A shared belief that team members can take risks, ask questions and admit mistakes without fear of punishment or humiliation.

Retrieval-augmented generation. A common grounding method: finding relevant information and supplying it to a model with each request.

Service level objective (SLO). An explicit target for a service's reliability or speed.

Smoke test. An experiment that presents a product or feature as available, to measure whether people try to use it before it is built.

Time to first value. How long it takes a new customer to see the outcome they bought the product for.

Unit economics. The revenue and costs associated with a single customer, used to judge whether growth makes the business better or worse.

Vanity metric. A number that looks good and goes up, but does not help you decide anything.

WCAG. The Web Content Accessibility Guidelines, the international standard for accessible digital products; Level AA is the common target.

Wizard of Oz test. An experiment in which people deliver a product's outcome by hand behind something that looks like real software.

L. Further reading

These are the books and studies cited in this playbook that repay reading in full, grouped by the part of the job they help with.

The product leader's job

- Marty Cagan, *Inspired: How to Create Tech Products Customers Love* (2nd ed., 2017). The clearest description of how strong product teams work, and the source of the four big risks.
- Teresa Torres, *Continuous Discovery Habits* (2021). A practical, weekly method for discovery, including the opportunity solution tree and the product trio.

Finding and validating problems

- Rob Fitzpatrick, *The Mom Test* (2013). A short, funny and indispensable guide to customer interviews that produce the truth.
- Clayton Christensen and colleagues, *Competing Against Luck* (2016). The jobs-to-be-done idea, explained by its best-known advocate.

Strategy, positioning and roadmaps

- Richard Rumelt, *Good Strategy Bad Strategy* (2011). Diagnosis, guiding policy and coherent action, and why most "strategies" are nothing of the kind.
- A.G. Lafley and Roger Martin, *Playing to Win* (2013). Where to play and how to win, from a company that practiced it.
- April Dunford, *Obviously Awesome* (2019). The most practical method for positioning a product.
- Joshua Seiden, *Outcomes Over Output* (2019). A short book that changes how teams plan.

The numbers

- Michael Marn and Robert Rosiello, "Managing Price, Gaining Profit," *Harvard Business Review* (1992). The classic case for taking pricing seriously.
- Mehrdad Baghai, Stephen Coley and David White, *The Alchemy of Growth* (1999). The three horizons of growth.

Teams and delivery

- Amy Edmondson, *The Fearless Organization* (2018). Psychological safety, from the researcher who defined it.
- Nicole Forsgren, Jez Humble and Gene Kim, *Accelerate* (2018). The evidence that delivery speed and stability go together.
- Donald Reinertsen, *The Principles of Product Development Flow* (2009). Cost of delay, batch size and queues, for readers who want the underlying economics.

Customers and design

- Christine Sinsky and colleagues, "Allocation of Physician Time in Ambulatory Practice," *Annals of Internal Medicine* (2016). A sobering picture of the work many healthcare products are meant to support.
- Ann Cavoukian, *Privacy by Design: The 7 Foundational Principles* (2009). Short, clear and still the foundation of privacy-respecting products.

AI

- National Institute of Standards and Technology, *AI Risk Management Framework* (2023). A structured, practical way to govern AI risk.
- McKinsey & Company, *The State of AI* (published yearly). A useful annual snapshot of how organizations are adopting AI and agents.