productboard

The Essential Guide to

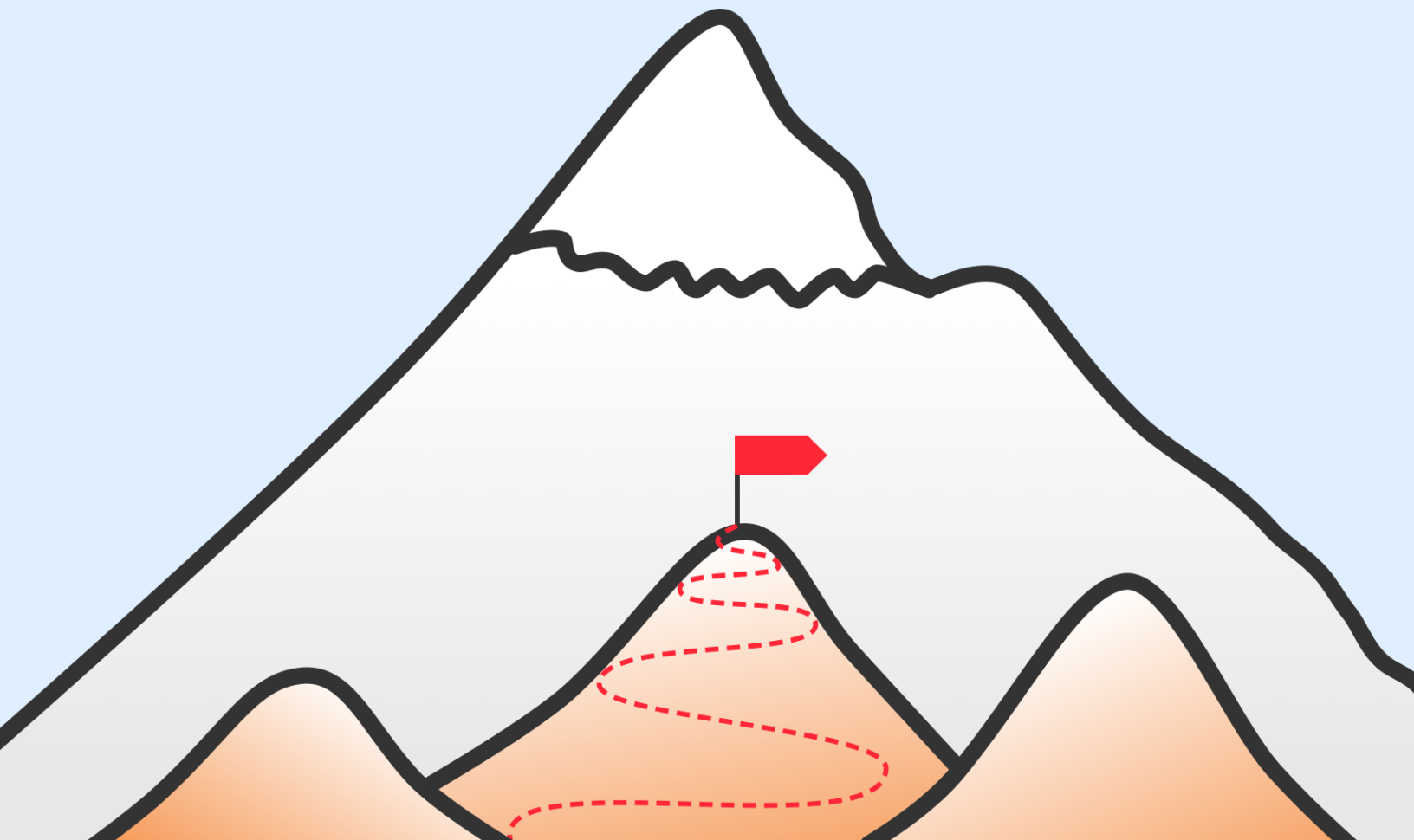# Product Discovery

Table of
Contents

productboard

Opening words

The discovery cycle is the process of actively capturing, researching, and prioritizing user needs—as well as collecting and validating solution ideas for addressing them. It complements the delivery cycle, which efficiently transforms validated product ideas into valuable product functionality.

Product discovery can be a touchy subject within the product world. While many of us know that it's important, it's easy to prioritize delivery and think discovery is a practice we can save for another day.

We have two goals for this eBook:

- To place product discovery in context: Why is it so important right now?
- To provide you with a playbook: How do you conduct better product discovery no matter where you are on your journey today?

Without further ado, let's get started.

The how and the what of product development

What's the point of optimizing the delivery of software if we're not delivering the right thing?

Quick! How many tools can you think of that help teams manage how software gets developed and delivered?

waffle.io ActiveCollab Clubhouse. planbox AGILE WORK INNOVATION Visual Studio Team Services

teamgrid Codetree Wrike REDMINE flexible project management

YouTrack sprintly FogBugz blossom ZenHub

ca technologies monday.com leankit HuBoard Podio

GitLab asana targetprocess teamwork
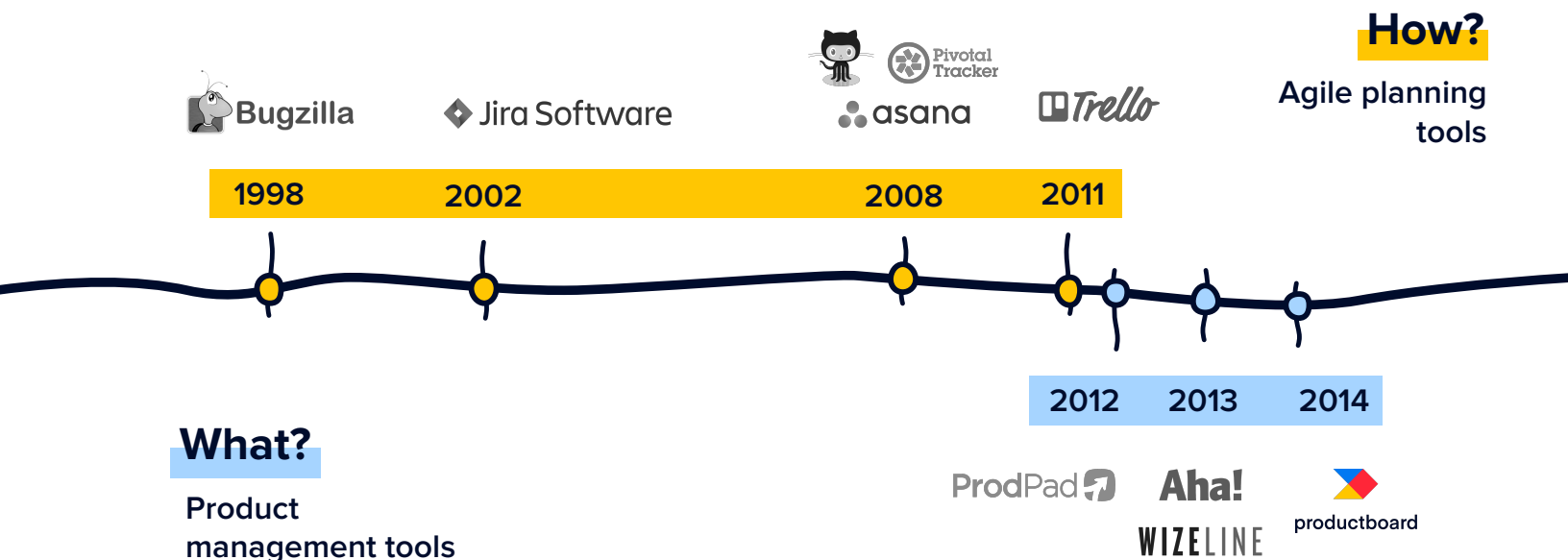
Basecamp Trello Jira Software Pivotal Tracker

Now how many can you think of that help not with managing how software gets built, but deciding what gets built in the first place?

Aha! ProdPad productboard WIZELINE

What's going on here?

Take a look at when the most popular tools in each space were launched

Bugzilla   ◆ Jira Software   asana   Trello   **How?**
Agile planning tools

| 1998 | 2002 | | 2008 | 2011 |

What?
Product management tools

| | | | 2012 | 2013 | 2014 |

ProdPad   Aha!   productboard
WIZELINE

It seems that four years ago, we all finally realized: What's the point of optimizing the delivery of software if we're not delivering the right thing?

But it's not that we never cared about deciding what to build. It's just that what has always lagged a little ways behind.

There's nothing new about the "what"

For over 25 years, we've had helpful frameworks for discussing *what* users need and *what* to build to address those needs. This is no new concept. It was back in 1991 when IDEO began popularizing the notion of design thinking—using rapid prototyping to iteratively uncover users' needs and design optimal solutions. Still, these methodologies were only used by isolated teams until more broadly adopted over a decade later.
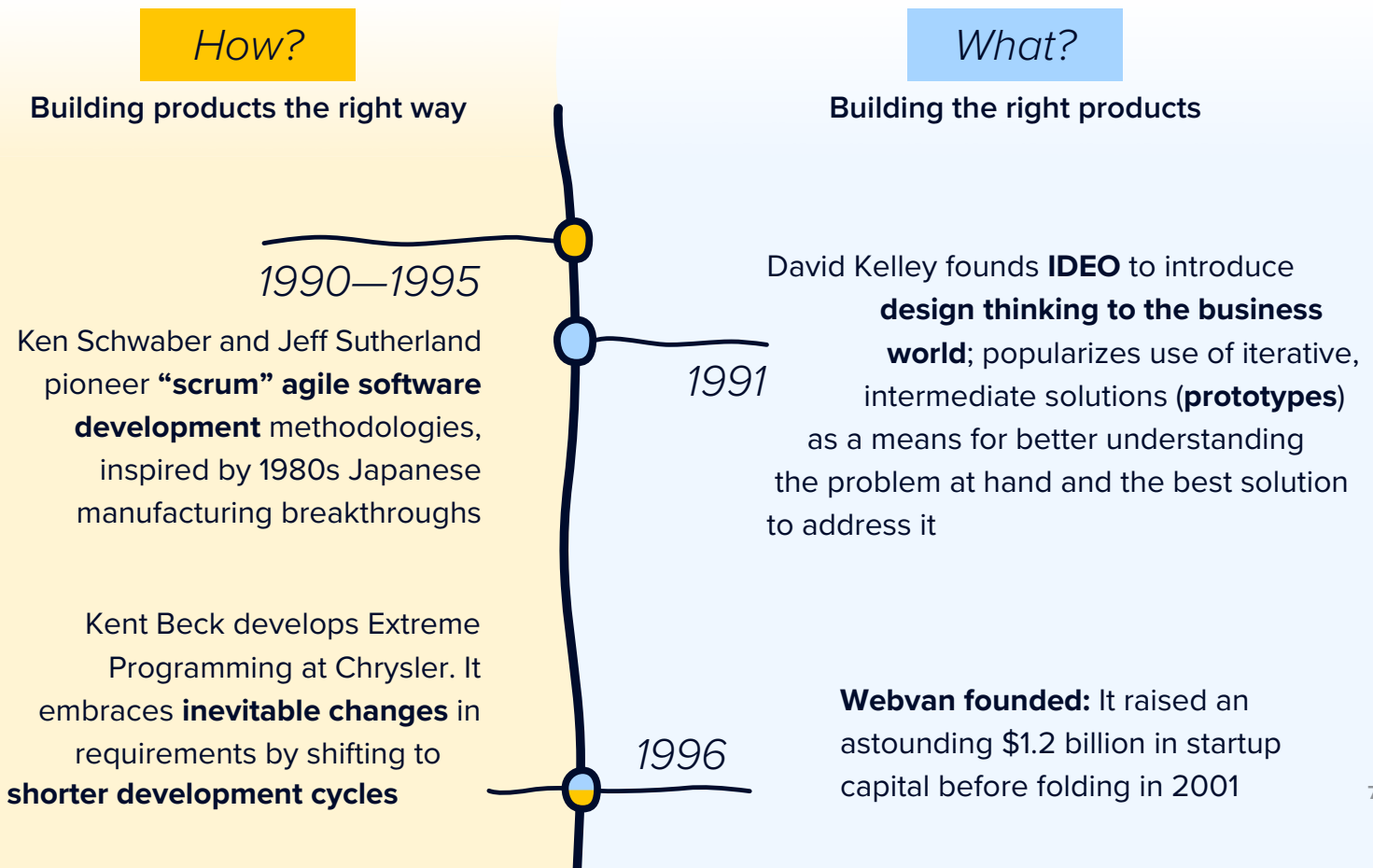
Indeed, the '90s were largely a time of optimizing marketing and engineering execution rather than systematically validating user needs. This came at a heavy cost. In the opening chapters of *The Four Steps to the Epiphany*, Steve Blank detail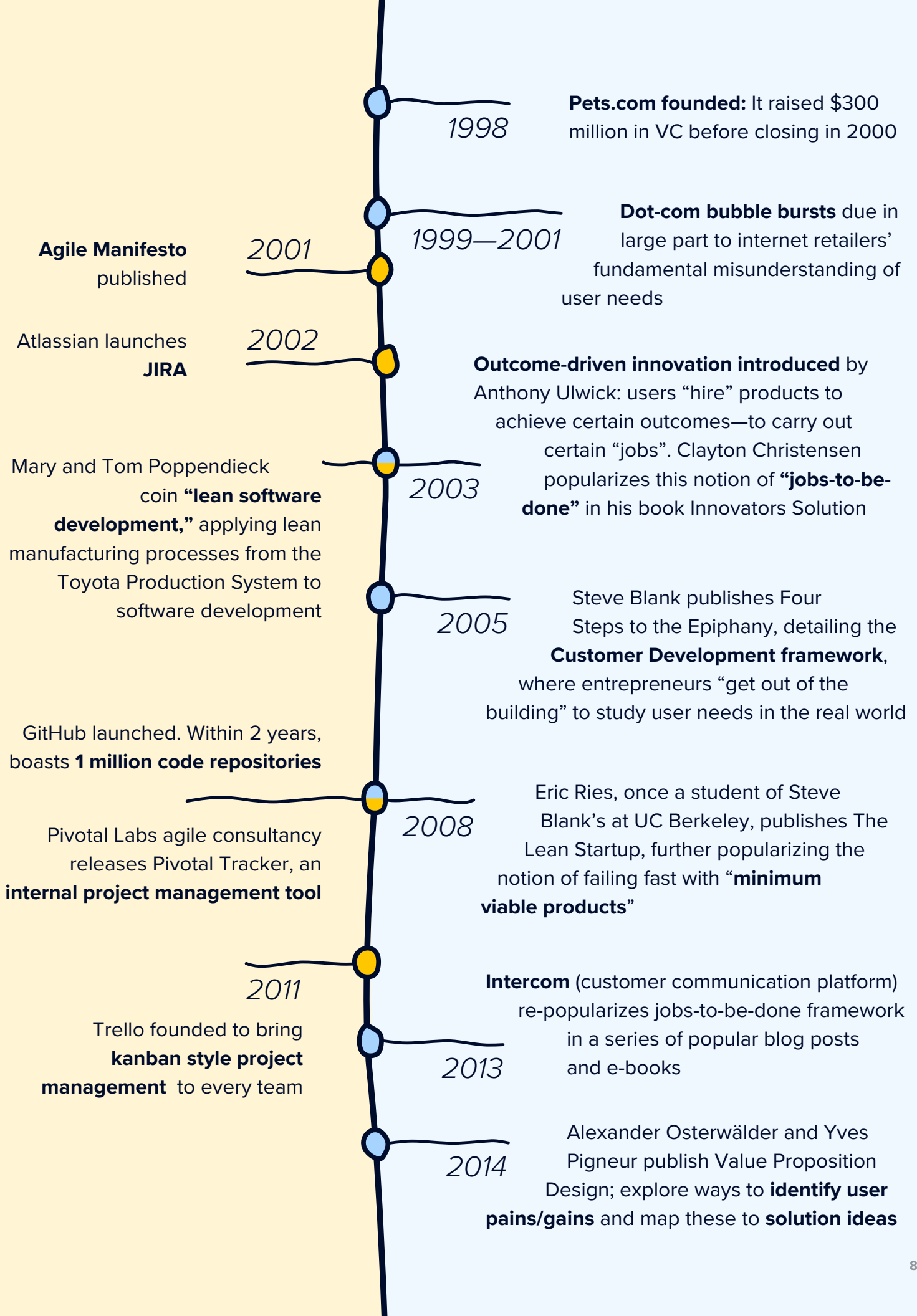s how WebVan, the early grocery delivery service, was so busy marketing *at* users and building hundreds of millions of dollars of infrastructure that they overlooked the fundamental needs of their target market.

$1.2 billion dollars later, they learned some of their early assumptions had been incorrect and filed for bankruptcy in 2001 at the height of the dot-com collapse.

And there wasn't just a disconnect between business models and user needs. It existed at the feature level as well. In *The Inmates Are Running the Asylum* (1999), Alan Cooper makes an urgent call to refashion the feature-centric view of product development, divorced of the needs features were meant to address.

His writings helped redefine the role of product managers and solidified user experience design as a field in its own right.

How?
Building products the right way

1990—1995

Ken Schwaber and Jeff Sutherland pioneer **"scrum" agile software development** methodologies, inspired by 1980s Japanese manufacturing breakthroughs

Kent Beck develops Extreme Programming at Chrysler. It embraces **inevitable changes** in requirements by shifting to **shorter development cycles**

What?
Building the right products

David Kelley founds **IDEO** to introduce **design thinking to the business world**; popularizes use of iterative, intermediate solutions (**prototypes**) as a means for better understanding the problem at hand and the best solution to address it

1991

1996

Webvan founded: It raised an astounding $1.2 billion in startup capital before folding in 2001

1998

Pets.com founded: It raised $300 million in VC before closing in 2000

1999—2001

Dot-com bubble bursts due in large part to internet retailers' fundamental misunderstanding of user needs

Agile Manifesto published

2001

2002

Atlassian launches **JIRA**

Outcome-driven innovation introduced by Anthony Ulwick: users "hire" products to achieve certain outcomes—to carry out certain "jobs". Clayton Christensen popularizes this notion of **"jobs-to-be-done"** in his book Innovators Solution

Mary and Tom Poppendieck coin **"lean software development,"** applying lean manufacturing processes from the Toyota Production System to software development

2003

2005

Steve Blank publishes Four Steps to the Epiphany, detailing the **Customer Development framework**, where entrepreneurs "get out of the building" to study user needs in the real world

GitHub launched. Within 2 years, boasts **1 million code repositories**

2008

Eric Ries, once a student of Steve Blank's at UC Berkeley, publishes The Lean Startup, further popularizing the notion of failing fast with "**minimum viable products**"

Pivotal Labs agile consultancy releases Pivotal Tracker, an **internal project management tool**

2011

Intercom (customer communication platform) re-popularizes jobs-to-be-done framework in a series of popular blog posts and e-books

Trello founded to bring **kanban style project management** to every team

2013

2014

Alexander Osterwälder and Yves Pigneur publish Value Proposition Design; explore ways to **identify user pains/gains** and map these to **solution ideas**

As it sorted through the ashes of the dot-com collapse, the tech industry was primed to reconsider the role of the product team. Who

Without dedicated systems for managing the what, teams are more susceptible to losing focus, building the pet features of the highest-paid executive in the room or being swayed by the demands of their largest, loudest customers—even if doing so is less likely to help their product succeed in the long run.

better to lead the industry's recovery and a new paradigm shift than the guy known for his theory that creative leaps come from destruction— Clayton Christensen.

In *Innovator's Solution* (2003) Christensen re-popularized the notion that users "hire" products to achieve some outcome, to get some "job" done. His jobs-to-be-done approach serves as a helpful framework for tailoring new features to users' underlying needs—whether you're making software or milkshakes.

In the years that followed, Steve Blank worked to apply many of the same ideas that inspired agile methodologies (on the *how* side) to the *what* side. One of the core tenets in his Customer Development framework (2005) was eliminating wasted effort (building features no one wants) by learning faster.

Eric Ries, a student of Blank's at UC Berkeley, channeled these ideas into *The Lean Startup*, expanding on Customer Development by detailing specific

Building shippable software is an expensive way to refine the understanding of what users need

practices startups (and any product team) can use to iteratively test hypotheses while pivoting from cheap prototypes to successful products.

More recently, we've seen a resurgence of interest in jobs-to-be-done with Intercom's popular blog posts and eBooks (2013–present).

Others have reframed jobs-to-be-done to further refine the way we discuss what users really need. In *Value Proposition Design* (2014), Alexander Osterwälder and Yves Pigneur describe a product's core value proposition as a function of the user pains it solves or the positive gains it provides. As they show, solving pains and delivering gains is the foundation of every business model.

"What" frameworks have advanced but the tools haven't

Returning full circle, the frameworks we now have for discussing and optimizing both the *how* and the *what* of product development have advanced considerably over the past two decades. But tools for managing each have advanced at different rates.

As an industry, we've favored the *how* to the detriment of the *what*. And that means that until now, product managers have managed the complex process of deciding what to build next using hacked-together spreadsheets, shared docs, task management tools, and systems for managing engineering, sales, marketing, and support data.

That's a shame, because it means teams have collectively wasted hundreds of thousands of hours and billions of dollars developing underutilized features and products that failed to live up to their potential.

Don't get us wrong. Optimizing the *how* hasn't all been a waste. Efficient software delivery is not inherently a bad thing. Being able to adapt to changing requirements, collaborate seamlessly, and improve continuously have led to higher quality software and more efficient, motivated teams. And because product increments are getting shipped sooner, teams collect feedback in less time and use it to optimize what to work on next.

The *how* and the *what* are interrelated, after all! All the same, building shippable software is an expensive way to refine your understanding of what users need. And even then, by limiting ourselves to learning from one product improvement at a time, we risk being blinded by the haze of incrementalism, unable to see if we're working on the optimal product or feature set to begin with.
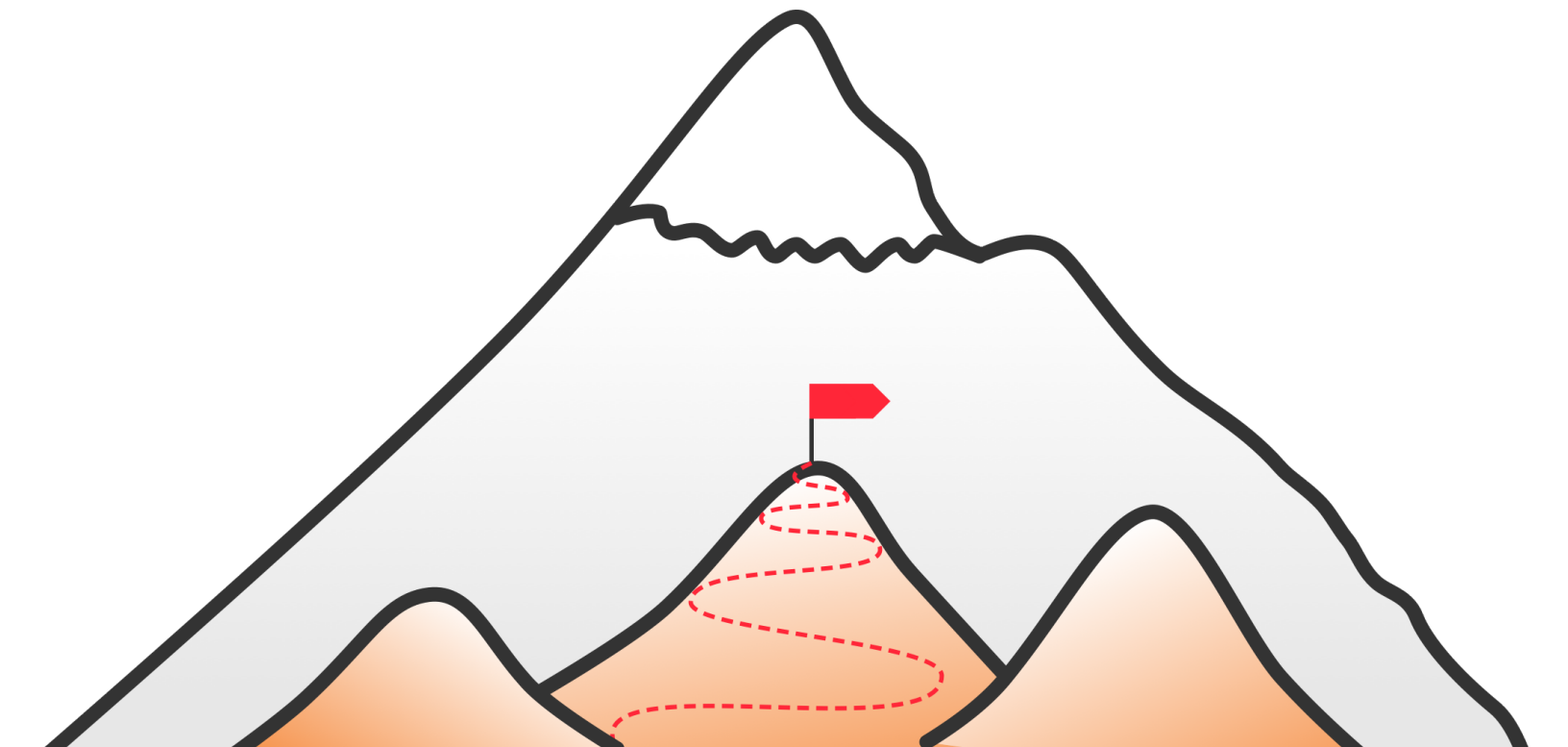
So how might we go about deciding *what* to build at the same time as we're optimizing *how* we build our products?
Head over to the next section to learn more.

The age of product discovery

Why risk building the wrong feature if you can first validate your understanding of the existing need and the ideal solution?

"There is nothing quite so useless as doing with great efficiency something that should not be done at all."

Peter Drucker

There is a better way to discover what users really need without slowing down the efficient systems we've developed for delivering software.

The answer is **dual track agile**.

Consider the typical agile product team, with product managers who:

- Process inputs like feedback and feature requests
- Generate feature ideas based on user needs and strategic considerations
- Prioritize the most valuable ideas, pushing them to the dev team's backlog
- Discard low-value or infeasible ideas

With just the top priority ideas on their plate, developers can deliver valuable features faster. That means feedback can be collected sooner, helping the product team further fine-tune *what* gets built.

But there's a problem.

At the very least, we're assuming user inputs can be neatly translated into requirements that product can send to the dev team's backlog.

Often, they can't.

Users have complex needs with subtleties *they* may not even be aware of. Product managers

who don't take the time to uncover them are doing one of two things:

- Making assumptions
- Encouraging developers to make assumptions (by passing them fuzzy requirements)

In either event, they're likely to get key details wrong, which threatens to waste the team's time and degrade the quality of the product. Building software, even efficiently, is an awfully expensive, frustrating, and suboptimal way of discovering what users really need.

- **Expensive** because software is time-consuming to code and maintain. Isn't there a faster, cheaper way to learn?
- **Frustrating** because developers don't enjoy committing to build potentially infeasible features with fuzzy requirements in tight timeframes, particularly when it might all be for naught
- **Suboptimal** because blindly iterating on an arbitrary feature idea until it's deemed valuable means not spending that time delivering another feature that *is* best aligned with the needs of your target market and overarching product strategy

In short, why risk investing in building the wrong feature if you can first validate your understanding of the existing need and the ideal solution?

By introducing product discovery to our process we can do just that:
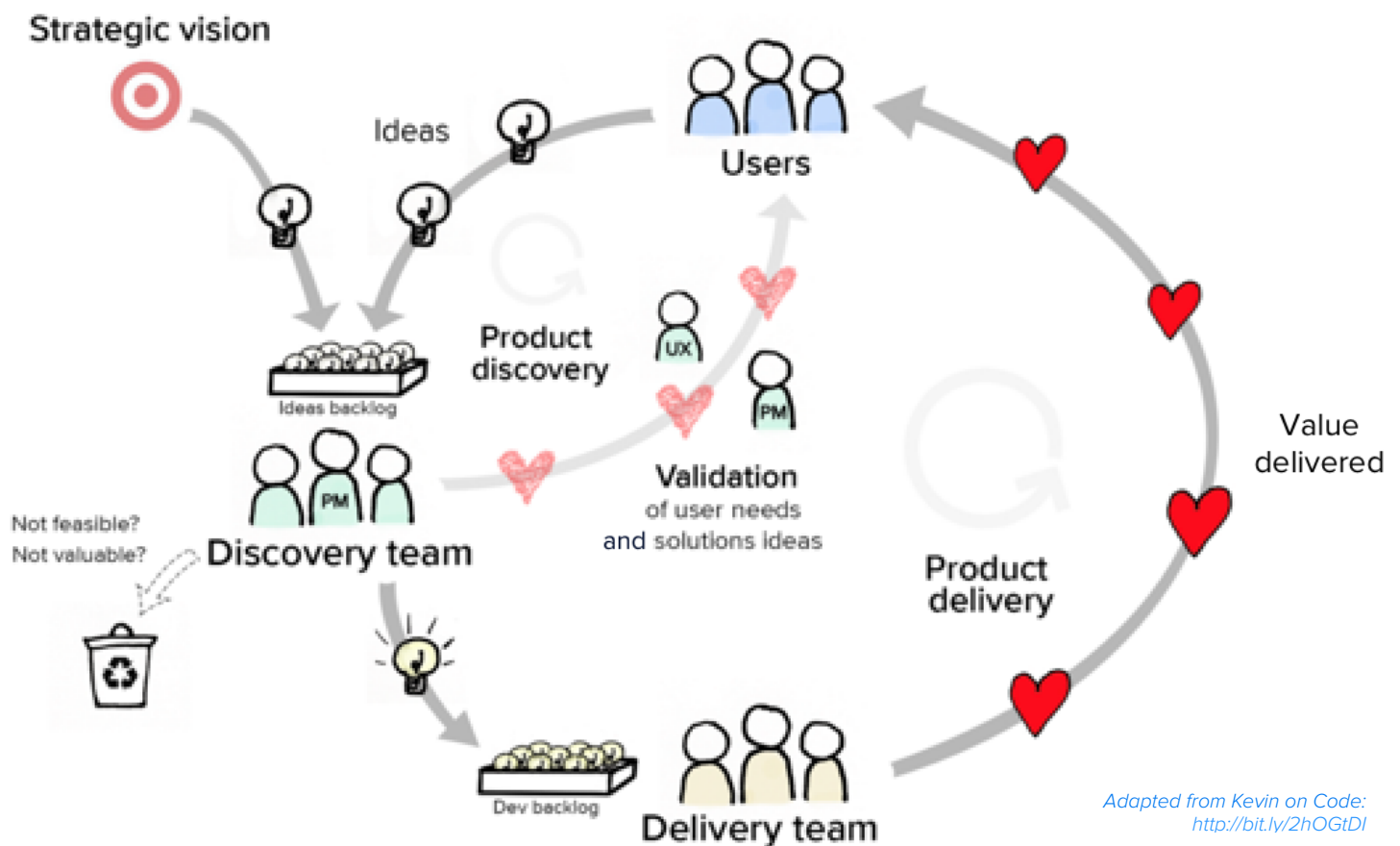
Adapted from Kevin on Code:
http://bit.ly/2hOGtDl

The discovery cycle is the process of actively capturing, researching, and prioritizing user needs—as well as collecting and validating solution ideas for addressing them. It complements the delivery cycle, which efficiently transforms validated product ideas into valuable product functionality.

At [productboard](#), our product discovery process begins while communicating with prospects and customers on [Intercom](#), over the phone, or in person during office visits and meet-ups.

"Hi there, I see you recently asked for a dashboard exports capability. Would you mind helping us understand why you're trying to export your data out of the tool?"

Customers routinely make requests or mention pain points they have with our product today, and our job is to dig deeper to understand what they are trying to do.

Product discovery can also be carried out more formally by interviewing users to delve into the context of their requests:

"We certainly appreciate you taking the time to screen share your existing system of hacked together spreadsheets. It's clear that prioritizing with a RICE score is critical for you. Mind if we take a screenshot to add to our research?"

With a clear understanding of the user's needs, PM and UX can team up to craft cheap prototypes and validate their understanding of optimal solutions for pressing problems. Ultimately, the aim is to understand:

"If we built feature X like this, would you use it?"

If feedback is positive, the prototype itself can serve as a feature spec when the feature is sent along to the delivery team, saving additional effort down the road.

Together, the discovery and delivery teams form a super team, responsible for bringing value to the customers. The super team works best when members from both teams collaborate to discover, validate, and deliver solution ideas.
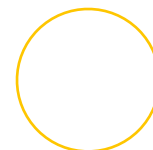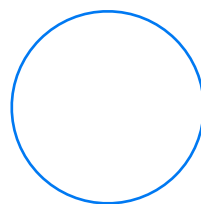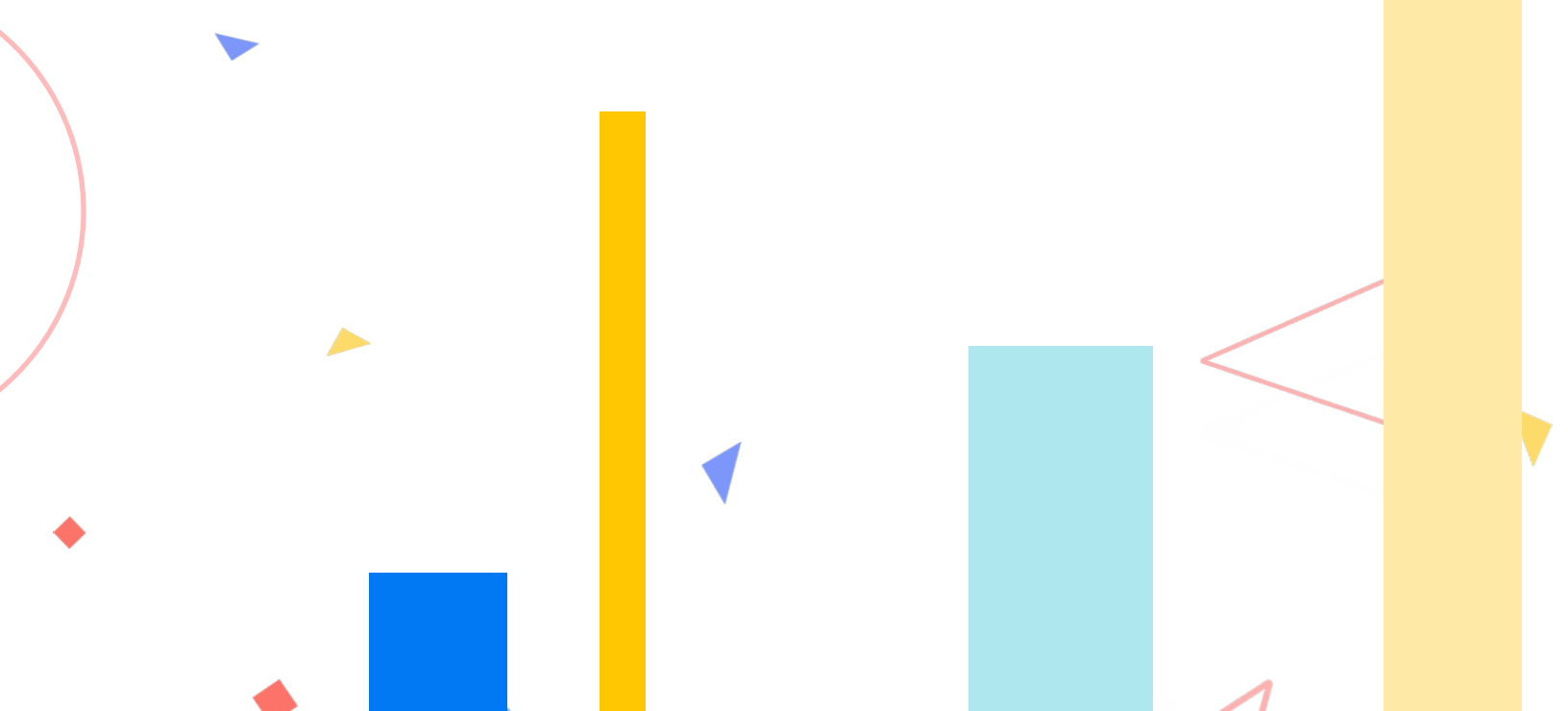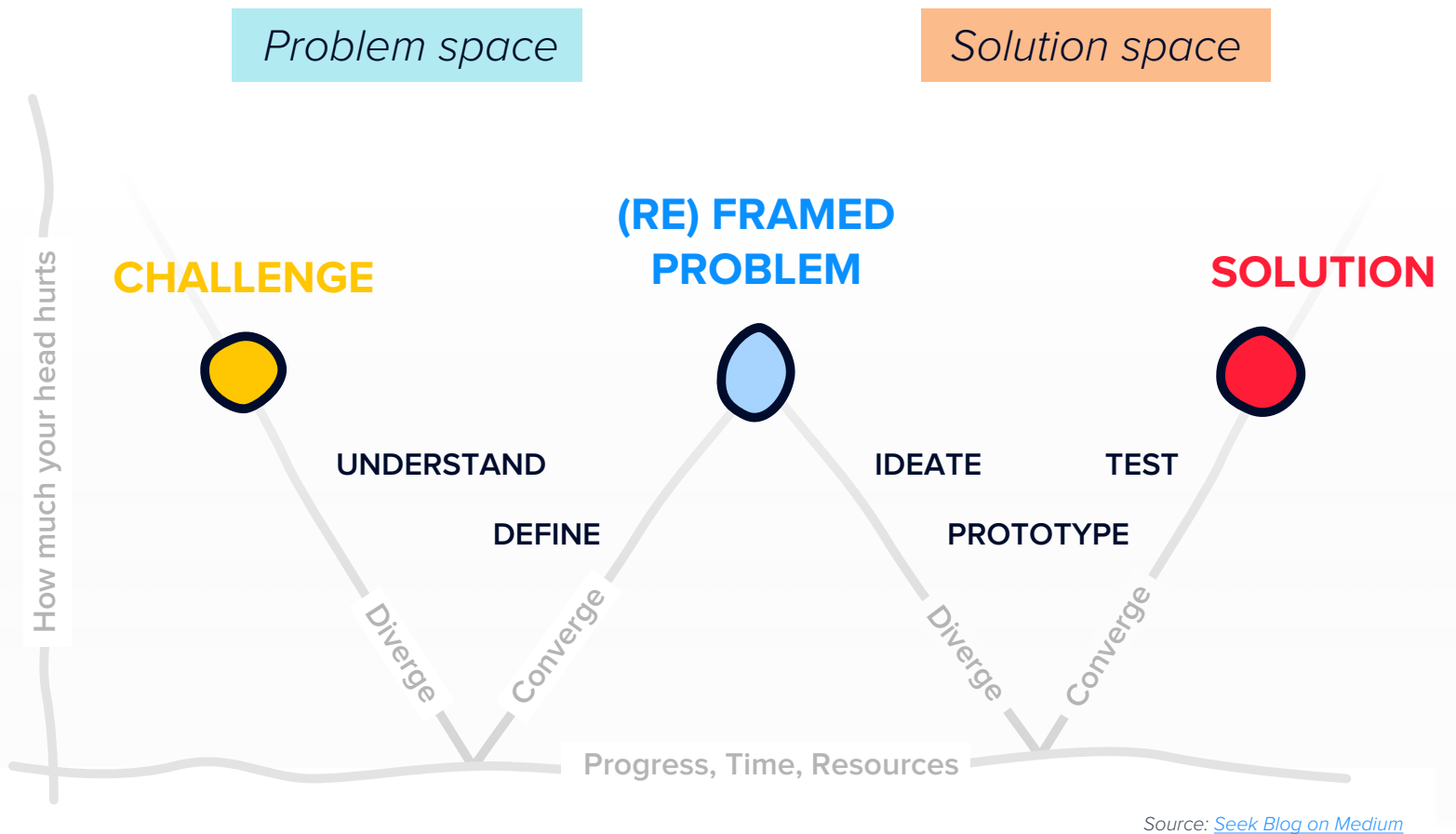
A step-by-step guide for conducting product discovery

Product discovery is about promoting an environment of learning so you can improve your product incrementally and consistently.

The Double Diamond is a popular approach for conducting product discovery

Problem space *Solution space*

How much your head hurts

CHALLENGE

(RE) FRAMED PROBLEM

SOLUTION

UNDERSTAND

IDEATE   TEST

DEFINE

PROTOTYPE

Diverge   Converge

Diverge   Converge

Progress, Time, Resources

Source: Seek Blog on Medium

1. Identify the challenge
2. Re-frame the problem
3. Present solutions

Let's break down each piece step by step

1

Identify challenges

In this phase:

Understand

Define

The product discovery process starts with identifying broad challenges that you are trying to solve with your product. This is the time for your product team to take a look at the big picture—high-level objectives or themes—not at specifics.

In productboard's case, a challenge could look like the following:

How can we enable mid-market companies to better use productboard?

Defining the right type of challenge is tricky. There are **new product challenges**, where you work on an open-ended blank slate. There are **value and need-oriented challenges**, which revolve around the current needs and pain-points of your users.

Then there are **growth vs. technical challenges**. Growth challenges are usually quantitative—maybe you are trying to improve a metric within your product, like user retention. Technical challenges are often related to product performance.

Understand

To correctly identify challenges, you must understand the underlying user needs you want to solve with your product. At this stage, product teams rely heavily on quantitative and qualitative research to find answers.

Ideas for tools and techniques that are useful:

- *User research*
- *Focus groups*
- *Observation*
- *Customer interviews*
- *Data analytics*
- *Competitive research*
- *Empathy mapping*

Define

After you understand the user needs you are trying to address, you must clearly define them:

- **Nail down the problem:** *Aim to nail down one sentence that covers the entire problem you are trying to solve. This helps you communicate clearly to your team and aligns them around a common cause. If you formulate the problem loosely, it will be difficult to keep everyone focused.*

- **Validate the problem:** *Make sure that you are actually working on problems that need to be solved. How big is the pain that your users are experiencing? How much value will tackling the pain truly add?*

- **Prioritize:** *You must then figure out which of the identified problems you should tackle first. There are several popular frameworks to choose from.*

Many product teams employ journey mapping, dive into the Five Whys or other similar techniques, or conduct a SWOT analysis for this step.

2

Re-frame the problem

In this phase:

*Ideate
Prototype
Test*

After you isolate specific user problems, it is best to reframe them into chunks that can be attainably solved.

For productboard, the broad challenge listed above can be re-framed and narrowed down into the following:

Mid-market companies experience limitations with productboard's public Portal because they want to communicate with multiple audiences at once.

During the reframing stage, you ideate, prototype, and test potential solution ideas that you have prioritized with your team. All this is due diligence to validate products and features before they go into delivery.

Ideate

You ideate to figure out how you plan to solve your users' problems.

This is where your team can get really creative with innovation exercises and other ideation techniques like

Team brainstorming

Mind mapping

Storyboarding

Running design sprints

After ideas are proposed, your team can gauge their potential impact and feasibility, then prioritize which to prototype and present to customers.

Prototype

[Prototypes enable teams to demonstrate their ideas](#) *and bring them to life.*

There are [many different types of prototypes](#), including but not limited to sketches, mockups, clickable prototypes, [MVPs](#), or even using [competitive/similar products](#). The type of prototypes teams choose to build depends on what they are trying to learn, what needs to be tested, and what open questions they still have.

Test

Testing determines whether or not the proposed solutions are actually capable of solving the problem. Popular tools and techniques here include: A/B testing, customer interviews, user testing, distributing surveys, and product beta testing.

3

Present solutions

Nothing is built yet at the solution stage, but you are ready to present ideas to stakeholders and users. Note that solutions don't necessarily equal features.

Going back to the example from productboard, here is a viable solution:

Let's enable customers to build multiple Portals so they can communicate with each of their audiences differently.

Getting to a solution can take numerous iterations, and presenting the solution to stakeholders (in productboard's case, product leadership, delivery teams, and cross-functional teams) will be crucial for earning buy-in and alignment.

At this stage, it is likely you will move into delivery, though not with a finalized design. Your solution is still rough around the edges.

Build excellence into your product discovery process

Here is a summary of how productboard evolved through this product discovery process:

Challenge

How can we enable mid-market companies to better use productboard?

Re-frame the problem

Mid-market companies experience limitations with productboard's public Portal because they want to share/validate their ideas with multiple audiences at once.

Identify a viable solution

Let's enable customers to build multiple Portals so they can share/validate their ideas with each of their audiences differently.

By using this framework and following its steps, product teams can create an environment of continuous learning that leads to truly excellent products.

3 mistakes to avoid with product discovery

3 mistakes that we often see in product discovery:

- *Jumping to solutions before nailing down real user problems*
- *Underestimating the importance of working with stakeholders*
- *Not involving engineers early enough*

Dive into details of each in the following pages and learn how to avoid them

1

Jumping to solutions before nailing down real user problems

<u>Understanding user problems</u> is the most important step when it comes to building a product. Yet, a common mistake product teams make is being so eager to <u>jump to a solution</u> that they don't nail down and define user problems first.

The average product team spends 80% of their time in the solution space and 20% in the problem space. Ideally, that time should be split 50-50.

Product thought leader <u>Rich Mironov</u> describes the consequences of this mistake nicely: "A lot of organizations skip user research, testing, and validation and get right into solutions and what they want to build. The result is a struggle to find a meaningful market that will use and pay for the product."

At productboard, the team aims to define exactly what they want to solve in a short, succinct statement—the more specific, the better. To accomplish this, it's often necessary to refine their understanding of the problem through several iterations.

Here are some examples of user problems the team has narrowed down while conducting discovery:

- Makers from multi-team organizations who process insights are reading notes that are not relevant to their product area
- Feature details of customers with multiple products [in productboard] contain fields that are not relevant to the feature. It makes it hard to navigate and creates an opportunity to make a mistake

2

Under-estimating the importance of working with stakeholders

It can be tempting for product teams to plow ahead with the discovery process without involving stakeholders from both in and outside the organization. This can cause big problems further down the road for a few reasons:

- **Product affects everyone.** Everyone's success—from top-level executives to individual contributors—is tied in one way or another to the product. Thus, it makes sense to consider diverse needs
- **Stakeholders have valuable insights to offer.** Stakeholders have knowledge that stretches beyond the product team's domain. Customer success and support, for example, interact with customers day-to-day. Sales is in constant contact with prospects and has an excellent pulse on the market. This expertise should be leveraged, not ignored
- **Product teams need stakeholder buy-in.** Product teams need the support of stakeholders to push through major product decisions. Including stakeholders from the get-go ensures that time and resources don't go to waste

Ideally, all stakeholders should have their needs considered during the discovery process so when it's time for delivery, they know why their request has been included (or not).

3

Not involving engineers early enough

Too often, engineers are only brought into the fold during the delivery phase. After all, that's when their skills are needed to turn ideas into real-life products and features. This is problematic for a number of reasons.

"If you're just using your engineers to code, you're only getting about half their value."

Marty Cagan

First, engineers won't fully understand the problem they are trying to solve because they didn't participate in the process of defining it. They may question why a problem was chosen in the first place, and product teams end up justifying their decisions rather than getting things done.

Second, product managers will be their only reference for understanding users. Thus, when it comes to identifying solutions, engineers may come to completely different conclusions than the product team because they did not play a hands-on role in getting to the bottom of user insights. Essentially, the product team will be two steps ahead and engineers two steps behind.

Third, the product teams risk missing out on great ideas if engineers aren't looped in. Engineers are often the smartest people in the room. They provide valuable technical insights, understand how to utilize existing functionality and reduce effort, and can help to quickly build a proof of concept.

Conclusion

Our goal with this guide is to give you both the theory and practice of product discovery. You now have an overview of how product management has evolved over time, a sense of why it's so important for companies to have a regular discovery practice, and the risks involved with not doing so.

We've also covered concrete steps you can take to begin building a regular discovery practice on your team and at your organization. This might feel like an intimidating process, but it doesn't have to be. Keep in mind that for many teams, adopting discovery is an ongoing process, so getting started is what matters most. You'll have plenty of opportunities to iterate and improve over time, but only if you take those first steps.

Thank you for reading

About productboard

productboard is a customer-driven product management system that empowers teams to get the right products to market, faster. It provides a complete solution for product teams to understand user needs, prioritize what to build next, align everyone on the roadmap, and engage with their customers. productboard is easy to use, enables company-wide collaboration, and integrates into existing workflows.

Learn more at productboard.com