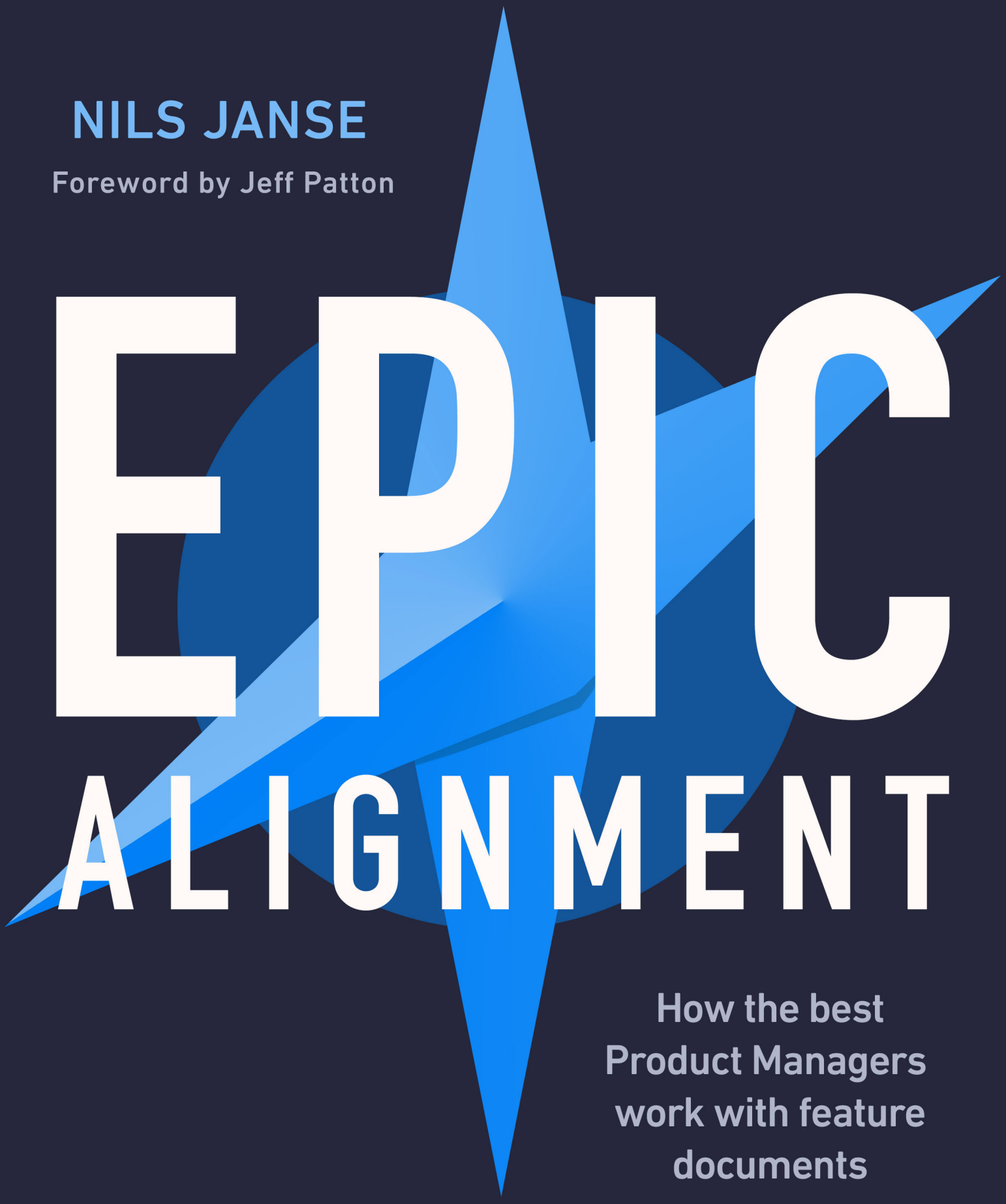
NILS JANSE

Foreword by Jeff Patton



How the best Product Managers work with feature documents

Said about the book

"Epic Alignment dives deep on one of the most important and underappreciated aspects of product management: using written documents to encourage collaboration and create alignment. With a good balance of general guidance and specific examples, this book provides valuable information for all product professionals, from those working on small teams with informal processes to those working at large enterprises with more formalized and extensive product development structures and frameworks."

- Matt LeMay, Author of Product Management in Practice & Partner at Sudden Compass

"Our product managers have been driving development work through structured feature documents for a couple of years now. Before that, we had really big problems with how to channel input from business stakeholders into the development work in a way that was not too disruptive. The approaches outlined in this book have really helped us at Storytel get on top of our whole product development process."

- Jonas Tellander, Founder and CEO at Storytel[1]

1 And also an early investor in Delibr

"As a developer, I have worked with a number of product managers over the years, and some of them left long-lasting impressions thanks to the quality of their work and collaboration with the team. When reading Epic Alignment, I realise that a common thread among those has been a focus on impact, and structured and collaborative decision making. I believe that product managers who take to heart the advice offered in this book would be a pleasure to work with."

- Johan Östling, System Developer and CEO at Citerus

"When we hold Certified Scrum Product Owner courses, we get to gather people and engage everybody in the same room with postit notes, legos and other physical props that pedagogically explain the main tenets of Agile product management. After the course, when the product managers get back to work, they often cannot gather everyone in the same room properly, and so need to translate the insights from the course into their day-to-day work environment. The approaches in this book provide a great bridge from Agile principles into everyday work."

- Reza Farhang, Partner, Educator and Agile Coach at Crisp

FOREWORD

By Jeff Patton

You've got a problem.

If you're reading this you're likely a product manager, product owner, someone trying to help one of these roles, or someone working to move into these roles. I'm not going to mince words here. You've chosen a tough path. You've got a lot you're responsible for, a lot to accomplish, and almost no authority to get it done. If you want help from others, it's going to take a mixture of wit, charisma, deep domain knowledge, strong business understanding, listening skills, storytelling skills, and luck. It's going to take a superhero.

But, if you're anything like me, you wouldn't have it any other way.

You're a bridge

You're a bridge between different constituencies with different concerns.

If your organization builds software, you'll work with architects, engineers, testers, devops and infrastructure people and likely others deeply focused on solving tough technical problems. They're stressed about what they need to accomplish and how little time they have to do it.

Your organization includes business stakeholders focused on keeping your business running smoothly. That's going to take everyone doing their jobs well and a steady stream of revenue from sales of your products or services. Business leaders are stressed about your business's ability to sustain itself, whether it can meet next quarters financial projections, or whether an unnoticed competitor or unseen economic event will hurt your organization.

If you sell a product or service, you've got customers who choose your product. They have other choices for how to solve their problems. They're stressed about making the right choice. They're worried that they're really getting value from the product they've chosen.

Finally, any product or service you build has users - the people that will actually type, click, speak, and swipe in the user interfaces you create. They're stressed about learning to use the product and actually being able to accomplish what they need to do using it.

Technologists, business people, customers, and users all stressed about very different things. You're not really any of those groups. Your biggest problem is to understand and address the needs of all of those groups. And, annoyingly, they don't care much about each other's needs. When a compromise is called for, no group wants to give up any ground.

Empathy, not authority

If you're a product manager, likely no one reports to you. You can't really insist anyone do anything. You can't tell engineers how to do their jobs or business stakeholders how to do theirs. You definitely can't tell customers and users what to do. Product managers manage products, not people. But, those products can't be built or evolve without the deep understanding and cooperation from every one of these groups. Sounds tough, right?

But, you have a perspective that others may not. You understand and empathize with every constituency's challenges, and how to frame things in a way they can understand. It's your superhuman listening and communication skills that will help you succeed here. If you don't have those yet, you will soon.

Even more importantly, you've got the ability to help each constituency understand and empathize with each other. You'll do this with lots of storytelling that helps them connect the dots between what's important to them and how it affects what's important to others. Creating empathy and understanding across groups is another superpower.

Alignment and trust

It won't be enough to empathize with each group, and help them empathize with each other; you'll need to get them all heading the same direction. You'll need to help them align around common goals. You'll need to help them understand how what they're doing contributes to that common goal and helps everyone. You can't insist people align. But, you'll feel it when you get alignment.

Something important happens when people believe you're on their side, when they believe you understand, and are fighting for, their concerns: they build trust. This is your secret weapon. All that listening and empathizing helps build it. So does deep your understanding of your product, your business, your market, and the technology used to build your product. You won't be able to insist people trust you. But, you'll feel it when you get their trust.

Building alignment and trust is another critical superpower.

Glube

It's not a very sexy superhero name, but it's the one I'm going to give you. It's a combination of two simple english words: glue and lubricant. You'll need to keep all these groups with different needs and interests glued together. But, they can't be stuck. You'll need to keep them lubricated - moving smoothly together.

Moving together is the only way any of them get what they need. It's the only way technologists can solve tough problems they're proud to solve, business people can sustain and grow businesses they're proud of, customers can choose products they're products they're proud of, and users can accomplish things with those products that they're proud of.

I'm hoping you appreciate the weight of the responsibility you hold. You've got lots of skills to build. You'll need lots more than you'll find in this book. But, this book will focus on a few of the most critical skills: the work you'll need to do to effectively communicate with, leverage input from, and keep all these different groups aligned and moving. You'll learn specific practices for building and maintaining shared understanding and alignment across these groups. You'll get references to other books and techniques to help you continue to build your superhuman skills.

You should get started right away.

SUMMARY

The book in four pages

How can you drive the thinking on what to develop, why and how to do it — leveraging the insights from both team and stakeholders — all the way from idea to deploy?

This book tries to answer the above question. I'm Nils Janse, the founder of Delibr, a startup that helps Product Managers. As part of our product development, we interviewed over 300 PMs/product people[2] to understand how they work and collaborate around feature development, what problems they face, and their approaches to solving those problems.

Some PMs are in a situation where they can have both team and stakeholders in the same room throughout the whole conversation on how to build a feature (e.g. those that work in small, co-located startups). That is the ideal situation, as then a shared understanding comes naturally. However, that is not the situation for most PMs (e.g. because they work in an organization with more than one PM, or with distributed team and stakeholders). This means they have to write down and communicate the relevant details. If you, like most PMs, are in the latter situation, then the insights in this book apply to you.

2 More than half agreed to be included in the Contributions section. Thanks for the help!

One of the main reasons why the PM job is so difficult is the total amount of complexity. For each feature, complexity comes from a combination of challenges across decisions (a lot of different, difficult, and interlinked decisions), people (differing perspectives, a mismatch between responsibility, expertise, and decision power), and process (rarely time to discuss, decisions often postponed or changed). This complexity then aggregates across many features and several PMs.

The scary thing with trying to handle this complexity by writing things down is that there is a risk that the amount of text written down makes it hard to cope with — facing a wall of 20 pages of unstructured text can paralyze the best of us[3]. The first chapter looks at how PMs struggle with this complexity, resulting in disconnected conversations and a lack of clarity on what to develop, why, and how to do it.

In our research, we saw four approaches help PMs succeed in creating clarity in the face of this complexity by making their thinking transparent. The following four chapters of the book explain those approaches.

> **1. Drive towards impact, base it on research.** When writing things down, the text can take a life of its own, making it hard to see the forest for all the trees[4]. That makes it extra important to know what the goal is. Chapter A describes how PMs can ensure

3 The book focuses on product management, but this is a problem that affects many entire organizations.

4 We are based in Sweden and so for us, this analogy makes a lot of sense.

they focus on the right things. PMs must be able to say 'no' to features. They can do this if they align with stakeholders on what business impact to go for and how it will be measured, show how feature outputs generate customer outcomes that in turn generate the business impact aligned upon, and do the research to be credible.

2. Embrace user stories as a shared language. The essence of structured problem-solving is breaking problems down into smaller parts. It is also necessary for making a written text possible to work with. There are many ways of breaking a feature down, and different stakeholders and team members will have different perspectives, even speak different languages. Chapter B shows how PMs can ensure customer focus and create a clear frame that everybody can relate to (so they don't have to translate back-and-forth). They can do this by working with user stories. This means moving from handing over strict requirements for features to build to having a conversation based on user stories. This conversation can be anchored to user stories not only early on with joint user story mapping sessions, but also later with details added to each user story.

3. Use a single source-of-truth to create shared understanding. One of the main scourges of writing things down is duplication. As soon as the same thing is written down in two places, the risk of misunderstandings and unnecessary work arises. Chapter C details how PMs can avoid this duplication and keep the documents structured and easy for all to interact with. They can

do this by capturing details in documents at the epic level[5]. These documents can then live across phases, based on a template that anticipates topics that come up.

4. Expose micro-decisions through explicit questions. A great benefit that comes from working with a structured document as the single source-of-truth is that you can know where any issue that arises fits. Chapter D shows how PMs can work in a more proactive way with decision-making, allowing them to bring in the team and come back to close down any loose ends. They can do this by recognizing that the work involves a lot of micro-decisions. These decisions can be written as answers to explicit questions within the relevant sections of documents, with decision status marked clearly.

Taken together, we have seen these four approaches help PMs both ship better products and save time for their teams and stakeholders. In the last chapter I show an example of what an epic document can look like, when working with the approaches. I hope you will find these insights helpful, and that they will help you become an even better PM!

5 The epic level encompasses several related user stories and either corresponds to a new feature or some change being made, but is neither a wide-ranging 6-month initiative nor a fixed area of the product.

Meet the complexities of Product Management

Before diving into the approaches that can help PMs create shared understanding around feature development, let's look at what the PM role entails, and why it is so hard.

The PM role

There are several quite good top-down ways to define the role of the PM, for example this one from the book "High Growth Handbook" that states that the PM is responsible for:

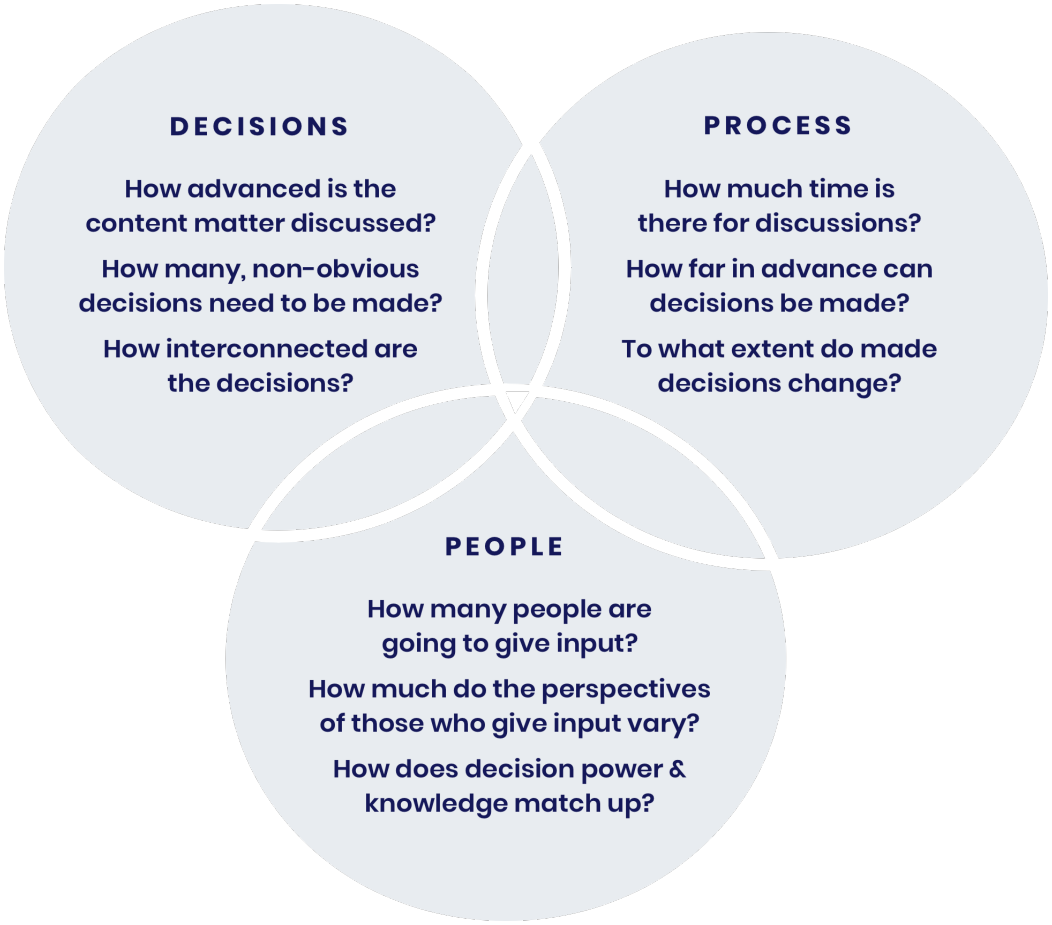
1. Product strategy and vision
2. Product prioritization and problem solving
3. Execution: timelines, resources, and removal of obstacles
4. Communication and collaboration (across 1-3)

I think it is quite good. But to understand why the role is so hard, we must dig deeper.

Why the PM job is so hard

When looking at the difficulty of any knowledge worker role, it is helpful to look at the "aggregated complexity" of the role. The job is to shape some kind of **decisions** through some kind of **process** in interaction with other **people**. The difficulty of the job relates to the aggregated complexity from these three factors.

To evaluate the aggregated complexity of a role, it helps to ask a few questions relating to the complexity coming from decisions, process, and people.



When looking at the PM role with that frame of mind, the PM job stands out even among most other knowledge work as very difficult. For an organization with a single PM working with one team, this might be manageable, but across several PMs and teams, the complexity aggregates and it is hard to find ways of working that scale. We see that almost all PMs in organizations with more than one PM struggle with this aggregated complexity.

Before we go into how to handle this complexity, let's look more in detail at the complexity that PMs face.

The archetypal PM process

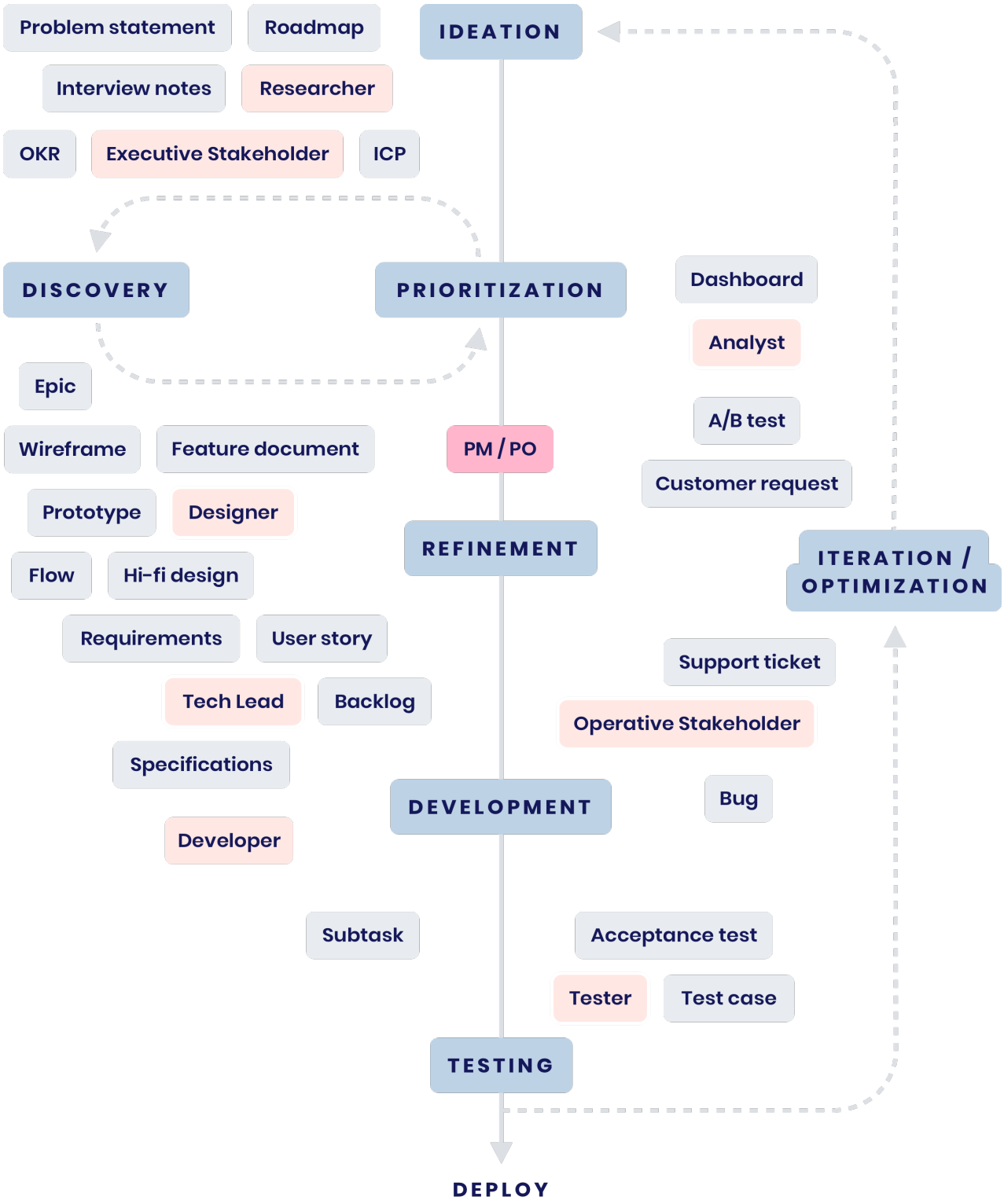
During our research, we tried to map out more in detail what it is that PMs actually do. To better understand how the complexity they face comes from people, process, and decisions, we looked at:

- **People:** What roles they interact with,
- **Process:** What artifacts, methods, & steps they go through
- **Decisions:** What type of decisions were made

Of course, all of this varies quite a lot between PMs and organizations, but as we talked to more and more PMs, and adapted our map, we started finding that almost all of the map applied to almost all teams. This was good enough for us to work with, and so we decided to call the map the "Archetypal Product Management Process", or the Archetypal Process for short. To put it simply:

- The **roles** involved are researchers, analysts, designers, developers, and testers, as well as different executive and operative stakeholders, with the PM at the center.
- PMs use many **artifacts** and **methods** (to mention a few: roadmap, user story mapping, feature document, wireframes, backlog, etc.) across the **steps** of feature development (that can be stated as ideation, prioritization, discovery, refinement, development, testing, and iteration/optimization).
- To drive this work forward PMs need to make, or at least facilitate, a myriad of **decisions** (taking into account the insights from team and stakeholders).

Here is a more detailed version of the map.



This may look at least somewhat straight forward, but one devious aspect of the work that almost becomes hidden when everything is laid out in neat boxes like this is the interconnectedness of the parts, and thus the iterative nature of the process. It is crucial for the PM to find disruptive factors and deal-breakers in e.g. design, engineering, and business, as early on as possible. That in turn is dependent on an understanding of the respective domains as well as working with experts to evaluate different options, with the goal of realizing what doesn't need to be decided early on but can be detailed later. Whenever this is not successful, and a deal-breaker is found out too late, the whole team may need to take several steps back.

How to not only survive, but thrive

Looking at this complexity across the decisions, process, and people for developing a single feature, and then thinking about how this complexity aggregates when developing many features, and working with other PMs doing the same, the resulting complexity becomes daunting.

However, although some of this complexity is real, a lot of it is merely complicated[6] and can be broken down, structured, and boxed in. By applying structured approaches of working, PMs can go from facing an impenetrable mess to working with a series of distinct and manageable parts.

6 Product development is inherently both complicated, in the sense that there are many moving parts, and complex, in the sense that the relationship between the parts and cause and effect can only be fully perceived in retrospect. Cynefin is a great framework for thinking about this by Dave Snowden.

As we interviewed more and more PMs, we saw that some PMs tended to adopt four such structured approaches that helped them to not only survive but thrive:

1. Drive towards impact, base it on research
2. Embrace user stories as a shared language
3. Use a single source-of-truth to create shared understanding
4. Expose micro-decisions through explicit questions

In the upcoming chapters, we will go through each approach.

CHAPTER 1

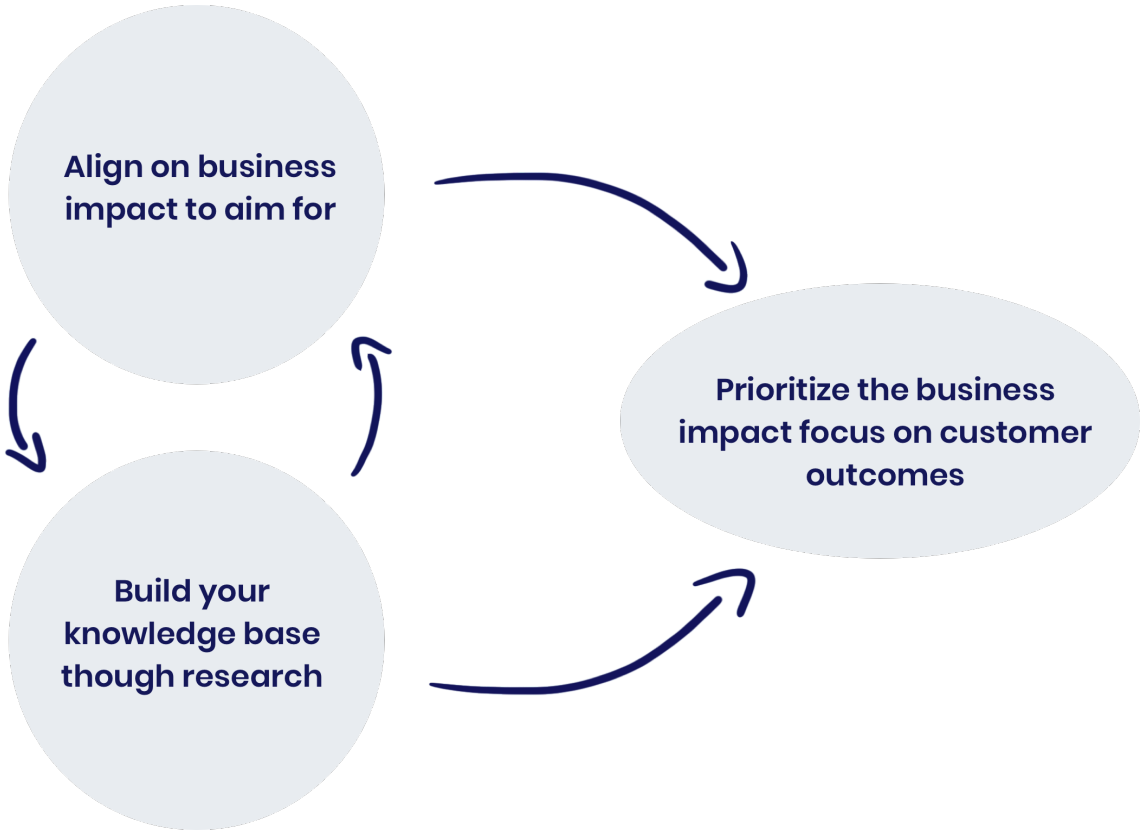
Drive towards impact, base it on research

Product Managers almost always have a lot of ideas in front of them. These ideas can come from customers, the team, stakeholders, research, or from just thinking hard. Most likely all of the above. If time was an endless resource, you could detail all those ideas into features and then build them all. Alas, time is scarce. Only a few things can be developed at a time. It is up to the PM to make sure this prioritization happens. Prioritizing means saying 'no' to some things.

Many PMs find it is hard to say 'no'. They face a lot of pressure from different parts of the organization, especially from stakeholders whose favorite idea did not make it onto the roadmap. In the worst case they feel forced to put more things in the roadmap than can actually be delivered, to placate the loudest stakeholders. This can lead to stress for the team and delays or rushed features needing rework.

Great PMs are able to overcome this pressure by doing three things, they:

1. Align on business impact
2. Build a knowledge base through research
3. Prioritize business impact, but focus on customer outcomes



We have seen PMs iterate around some version of these steps, and as a result, because this gives them something of an anchor, they are more comfortable with cutting through the complexities and actually saying 'no'. Let's go through more concretely how to do this.

1. Align on business impact

Good PMs drive towards impact. Stephen Covey put it nicely when he said "Begin with the end in mind"[7]. This is a good habit in general, and so also for PMs. Having an impact to aim for clearly defined and agreed upon makes life as a PM way easier.

Think about impact like top management

A mistake that many PMs do is that they fail to really take a step back when they think about impact. Increased conversion at a certain step, or resolving a specific customer request is great, but it is not certain that executives will have the time to translate that into impact that is meaningful for them.

The best PMs translate the impact into something meaningful for the executive stakeholders in advance. The way to do this is to think like top management, and to do the work of tracing the impact of any improvement all the way up to a level where it is directly meaningful for them.

7 "Begin with the end in mind" is one of The 7 Habits of Highly Effective People, a book by Stephen Covey. It is a great book on personal leadership that I'd recommend all PMs to read!

A common framework for translating potential product improvements into something that top management can absorb is A3R3 or “Pirate metrics” (as it can also be written out as “AAARRR!”, which sounds like something a pirate would say).

The idea of Pirate Metrics is to optimize each level of the conversion funnel, ultimately resulting in revenue. One of the benefits of this framework is that it is a good middle ground between how PMs think and how executives think. All the six steps are relevant to both.

There are three notable exceptions to this:

- If top management has another clear and communicated framework for how to quantify impact, then it probably makes sense to use that framework.

- If it is explicit that revenue is not the main goal, then it is often better to use a framework that emphasizes something like active usage or engagement[8].
- If the product being worked on does not have clear external customers that go through funnel steps similar to AAARRR, then it is better to try to think of and agree on another framework.

If none of the exceptions apply, then the Pirate Metrics are a good starting point.

"Having a clear product vision here and objectives is key. If you can clearly trace your product prioritization decisions back to what the company is trying to accomplish (ideally backed with data), it makes your job to convince all stakeholders a lot easier. In most cases, the Execs were the ones who came up with the vision or at least agreed to it in the first place. That said, they should have visibility into this prioritization process (not to be done in a vacuum)."

- Hiram Vazquez, Product Manager at ReloQuest

8 Mixpanel's Guide to product metrics elaborates on how PMs can find a good framework.

Define impact clearly

The first half of having good, impact-based goals to align on with executives is linking them to something they can relate to (e.g. one of the Pirate Metrics). The second half is breaking them down into something that is concrete and measurable for you as a PM.

OKRs are a common format breaking down goals and making them measurable. OKR stands for Objectives and Key Results.

> **Objectives** are high-level, aspirational, and qualitative. They are something that you aim to achieve, but should not contain numbers.
>
> **Key Results** on the other hand are a bit more low-level, attainable, and quantitative. All key results are linked to an objective, and each objective typically has around three key results. Key results should be measurable, and formulated in a way so that it is unambiguous whether they have been met.

Whether you use OKRs or something else, the important thing is to align with your executive stakeholders not only on what impact to achieve but also how to measure that impact. That is what enables you to take ownership of prioritizing what to build.

"Consider how in the early stages you need alignment on what the problems you're looking to solve are; sometimes the business impact is quantified from different areas, e.g. optimizing part-time employee schedules might lower attrition (biz impact #1) and reduce over-staffing (biz impact #2). Unless you have alignment on which goal to go for, you can end up working in the wrong direction."

- Elisabeth Kamor, Product Manager at Amazon

Master upward management

Like we talked about, an important aspect of being a good PM is being able to say 'no'.

Most PMs work with some kind of executives. In the ideal case, they support the PM and back up the PM's decisions. This type of support can be invaluable throughout the development process. However, PMs also face the risk of being overruled by upper management. That is why "upward management" (or managing up) is such an important concept.

It is easier to get acceptance from executives when you say 'no' to something if they have previously agreed to the premises you use in your rationale for saying 'no'. Still, an executive who sits in a meeting and is surprised by a 'no' to a feature they want to champion are more likely to try to overrule that decision than if they were informed in advance.

That is why it is important to not only align on what impact to go for, but also to keep in mind who might object, and to keep them in the loop.

> *"Many PMs miss the importance of managing up. And then they end up in a situation where they will have to do re-work, wasting the time of the team."*
>
> ***- Matt Lerner, Lead Product Manager at Mindbody***

2. Build a knowledge base through research

Good PMs know what they talk about when they talk about how to achieve impact[9]. To get there, they need to do their homework.

Link your research to impact

You need to know your customers inside-out. But PMs always have difficulties setting aside enough time for research, with endless meetings, competing priorities, and a constant lack of time. To make the time you have count and to avoid trying to boil the ocean, you need to focus.

9 As Haruki Murakami might have said, had he written a book about product management rather than about running...

To focus, and make the research relevant, the research must address three questions consecutively:

1. What business impact is most important to achieve?
2. What improved customer outcomes would achieve business impact?
3. How would different development efforts improve customer outcomes?

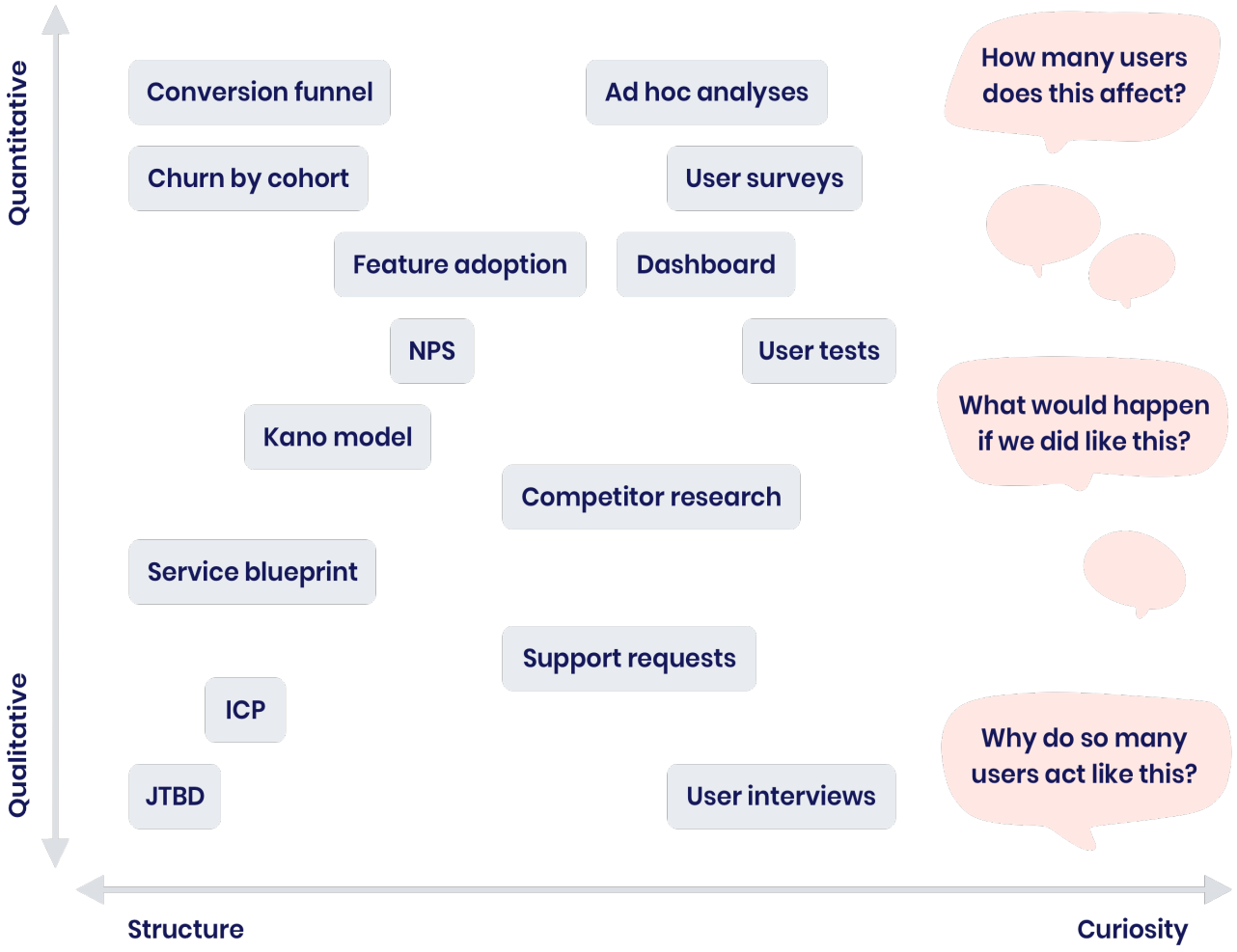
By answering these questions over several iterations, it is possible to do more effective research. If you don't do research, your answers to these questions won't be good. If you don't ask these questions, your research won't be useful.

Conduct different types of research

To know your customers inside-out you need to approach your research from several different angles. For every type of research, throughout the different stages of maturity of the product you are working on, at a certain point, doing more of it starts to yield fewer and fewer insights. But coming at the problem from a different angle can typically yield new insights. It is possible to get more insights out of research efforts by being aware of the diminishing returns to every type of research.

There are many ways to distinguish different types of research. We have found the two perspectives of structure vs. curiosity and

quantitative vs. qualitative to be a helpful way of mapping different research efforts. By making sure to do research from different parts of the map below, PMs can avoid falling into the trap of doing research that does not yield more actionable insights.



The distinction between quantitative and qualitative research is the most obvious. With only quantitative research, there is a risk that the data is being interpreted in the wrong way. With only qualitative research, there is always the risk that the insights will be (or be seen as) “fluffy” and not substantive enough to take action, or even worse,

will be anecdotal and misleading as a basis for decision making. To simplify, one could say that you need to put on the hat of both the UX researcher and BI analyst. This can either mean actually conducting such work yourself, or, even better if possible, getting more mileage out of your scarce time by working closely with people in such roles. For doing research, User Interviews can source candidates for you. The scheduling software Calendly helps you meet with more users by removing the messy back-and-forth scheduling emails—your candidates can self-schedule by directly booking predetermined slots in your calendar. For doing analyses, Segment can identify where users come from and track what they do in a standardized way, and then send that to Mixpanel where product teams can complete the relevant analyses to understand both how these users engage and convert as well as where they churn or experience friction, in real-time across devices.

The distinction between structure and curiosity is perhaps a bit more subtle. There are a bunch of great frameworks for doing different kinds of research, some of them captured above. The main benefit with frameworks is that they provide structure for the research, covering different important topics. The main risk with relying too much on these structured frameworks is that research can become an exercise in filling out a template, without asking critical questions. Without those questions asked there is a risk that the research won't really yield any insights. Get more relevant insights out of your research by switching back-and-forth between working with structured frameworks and taking a step back and asking critical

questions like “why..?”, “how..?”, and “what if?”[10].

“For our early customer interviews, we developed structured interview guides with clear objective statements for each conversation. However, it is important to enter every conversation knowing that the interview guide is only a “guide.” Lead every conversation with curiosity and know that it is okay to deviate from the structure you put in place (just remember to take things back if they get too off track). The customer is the expert of their domain, so talking to them is an opportunity to learn from them. Be curious!”

- Jafar Owainati, Co-founder at Loopio

Combine your research into a knowledge base

One great thing with research is how it accumulates over time. Each analysis, prototype test, interview, and survey has the potential to build on previous research. But for this to actually happen, the PM needs to codify conducted research, and make sure it is structured in a way that makes it easy to access.

Here are a couple of examples of benefits from having a combined knowledge base:

- When conducting an interview targeted at a specific problem, that interview can be more targeted if the interviewer has

10 The five whys is a good technique to encourage asking questions.

access to previous interviews and analyses relating to the problem.

- When writing a survey, it is best to base it on what has been said in interviews, and conversely, it is good to have any existing survey replies from a user handy when interviewing that user.
- When testing a prototype of an upcoming feature, it is more fruitful to do it on someone that previously reported the problem which the feature is trying to solve.
- When working on your Ideal Customer Profile, it is good to both look at interviews with a few concrete customers, and look at analyses on conversions based on more generic properties.

Some PMs are diligent about building a Wikipedia-like knowledge base, where they tag and link interviews, surveys, analyses, etc. The PMs who do this, and also do the day-to-day work of advocating for research and connecting it with customer outcomes, tend to be able to get the benefits outlined in the above examples. The PMs who don't do something like this face a tendency for research to be limited to what is in the head of the PM or the main researcher, and so many of these benefits are lost.

"Ultimately, a holistic understanding of the user helps us make smarter product and growth decisions to innovate faster and improve the customer experience."

- Pranav Kashyap, Group Product Manager at Mixpanel

3. Prioritize business impact, but focus on customer outcomes

Ideally, by now, you have aligned on the business impact to go for and built a knowledge base through research to back that up. But to get any value from this you also need to translate it into what your team should work on. Here we have seen PMs succeed by doing four things:

1. Differentiate between output, outcomes, and impact
2. Prioritize business impact
3. Translate down to feature outputs
4. Focus actual work on customer outcomes

Distinguish between output, outcome & impact

Let's look more closely at the terms output, outcome, and impact[11]:

Feature output is what developers produce, e.g.

- user story
- A/B test
- improvement
- bug fix

Customer outcomes are the benefits to the customer, e.g.

- problem solved
- behaviour changed
- improved app rating

Business impact is the benefit to your own organization, e.g.

- problem solved
- behaviour changed
- improved app rating

We have seen much confusion arise for PMs when the distinction between these three terms has not been clear.

11 The distinction between the three is also well explained here by Christophe Achouiantz, Agile consultant at Crisp. And, no, sadly, "tristinction" is not a word.

"My primary responsibility as a PM is to answer the question of why we develop features by linking them to customer outcomes and then on to business impact. Together with the team we detail what feature output we could do to reach sought outcomes and impact. Here we test different things until we find the right ones, and I contribute as a PM, but expect the whole team to be engaged."

- Johan Bouzi, Product Manager at Avanza

Prioritize impact and translate to output

In Scrum, the goal of the PM[12] is to allocate the output of the team to get the most customer outcomes, i.e. to maximize outcome/output. This is a good but incomplete description of the goal. It is too simplistic, both because business impact and customer outcomes sometimes diverge and because PMs must often communicate with stakeholders that are mainly interested in business impact.

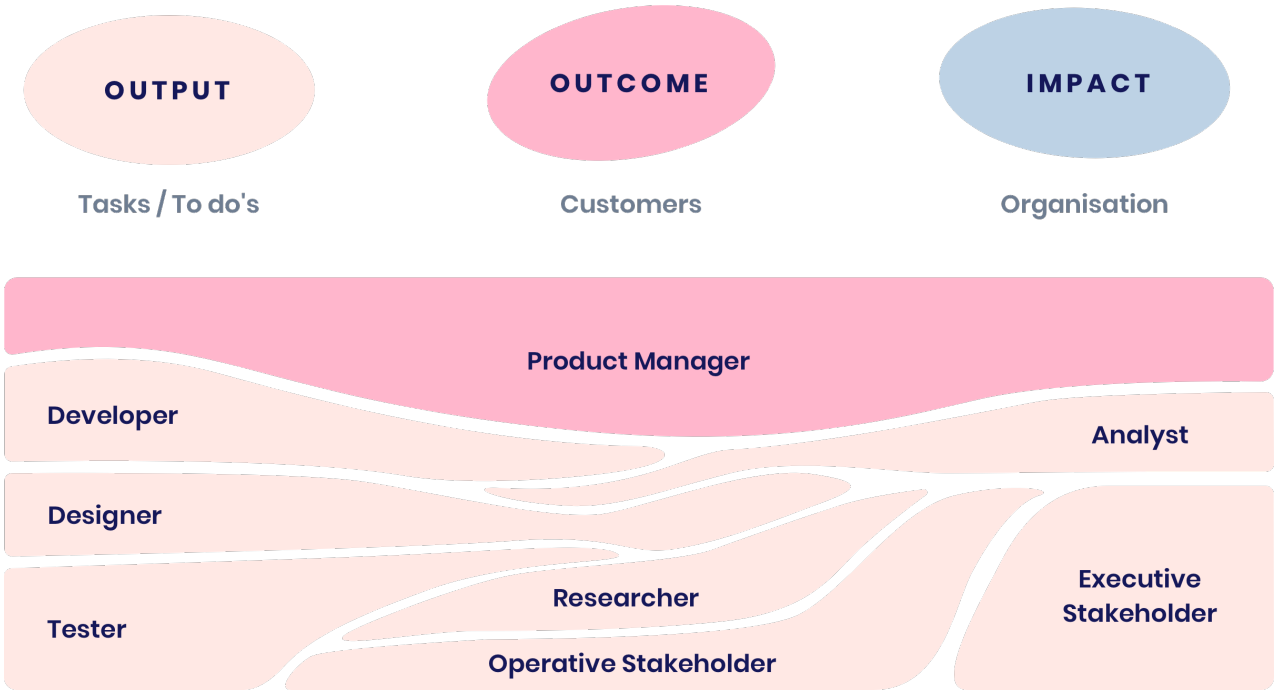
Let's look at an example of when impact & outcomes are not the same:

> *Say you have a critical business challenge in getting new users through your onboarding flow (part of Activation, the third A of the Pirate Metrics). You have a few, quite happy customers, but almost all new users drop off right after signup. Then it is probably not the right priority to direct the team to develop the next new feature*

12 Or actually, Scrum uses the term "Product Owner" (PO). There are many views on the distinction between the two terms. We tend to believe that the best view is that PM is a job title and PO is a role in a team and that they should ideally be the same person.

> *your existing customers have been asking for (which might improve Retention, the first R of the Pirate Metrics). Instead, you must prioritize the customer outcomes that drive the sought business impact, i.e., in this case, you must work to improve onboarding.*

Let's also look at the roles from the Archetypal Product Management Process and what focus they typically have across output/outcome/impact:



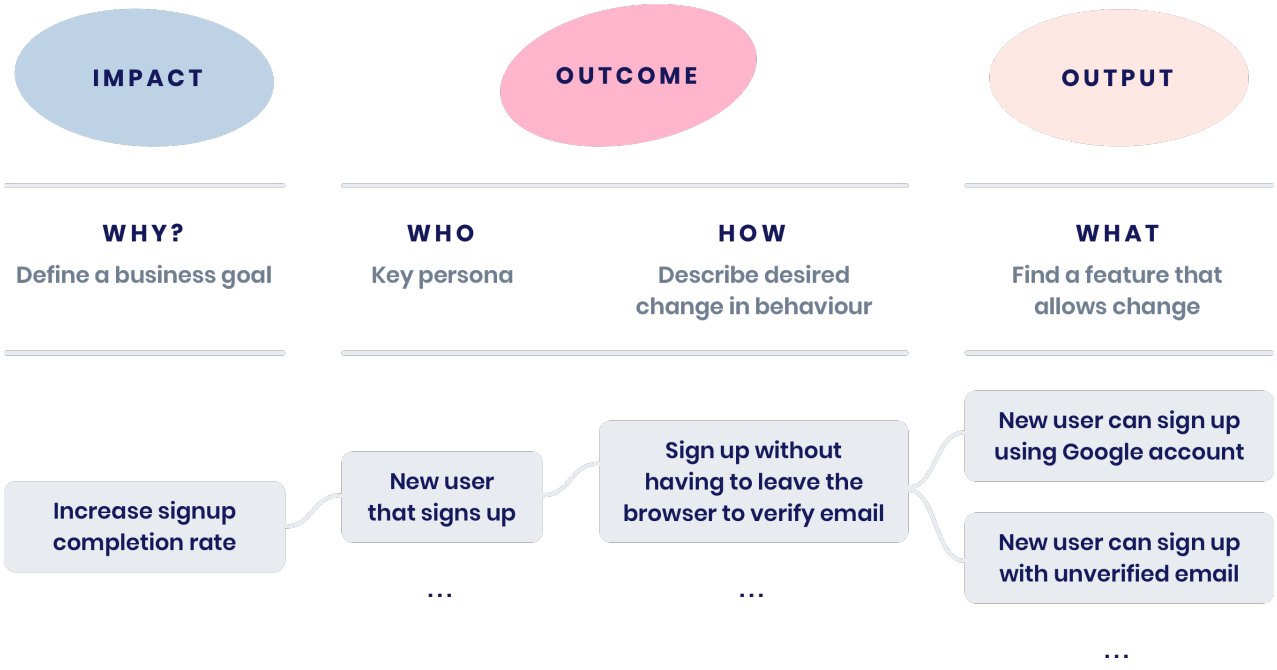
The PM has to work across output, outcome, and impact. Since executive stakeholders often have significant decision power and are often focused mainly on business impact, it becomes important to articulate that link. Not only that, developers and designers also often have a lot of decision power. But even though the PM works across the different aspects, the PM is not the CEO of the product. In fact, it

is rare that the PM has full decision power. Instead, PMs must learn to wield their "soft power" and make sure they find good solutions that they can get the team and stakeholders to align around.

Thus, the goal of the PM should be to maximize impact/output, in theory extending the formula to "impact/output = impact/outcome * outcome/output". It will be important to link business impact to customer outcomes and all the way down to feature outputs, as well as to estimate the effort required to build feature outputs.

Impact mapping is a good method for this linking that PMs can use whenever a trade-off has to be made between different features. We have seen different variants used. Sometimes we have seen feature outputs (deliverables) directly linked to business impact. Such links rely on far-reaching assumptions, as the benefit to the user is implicit[13]. Below is an example of an impact map that explicitly links business impact to customer outcomes and then on to feature outputs:

13 Here is a good explanation on the risk of losing sight of customer outcomes when doing impact mapping by Kurt Bittner.



However, it is not possible to prioritize doing something without taking the required feature output into consideration (the lower part of the equation). Feature outputs need to be estimated before any delivery work begins[14]. If something would take an inordinate amount of time to build, try to find something else. It means the PM needs to talk to the team to figure out how hard things are to do.

By combining impact mapping with doing estimates, PMs can better prioritize what to develop and become empowered to say 'no'. They can then protect their development teams from being handed work that does not have impact.

14 Although it is both possible and desirable to down-prioritize even estimating feature outputs that don't provide any value. Estimation also requires effort.

"Mapping feature outputs all the way to business impact allows me to say no! - if it is not clear how it drives impact, it probably does not belong."

- Chrysanthos Nicholas, Product Manager at Litify

Focus actual work on customer outcomes

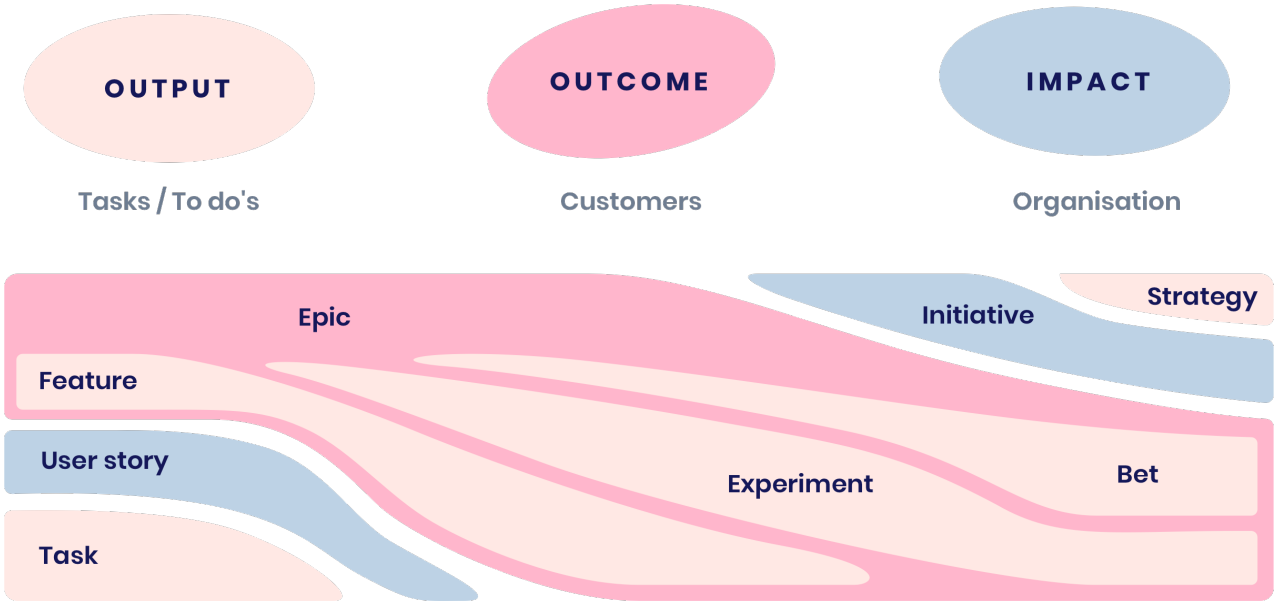
If the formula is "impact/output = impact/outcome * outcome/output" and the goal is to maximize impact/output, then it could seem like PMs could simplify things and not think too much about customer outcomes. But therein lies a trap, as PMs that only focus on business impact and feature outcomes risk not paying enough attention to customers.

It is important to make sure the targeted customer outcomes support the sought business impact. But too much focus on impact brings risks of making a series of short-term improvements rather than actual customer innovations. It is also less inspiring for the team, as great teams derive meaning from helping their customers. Ideally business goals and customer needs are aligned early on in the process, resulting in a positive flywheel effect, as if they are perceived to be in conflict, it is likely a signal of some underlying problem.

It is also important to make sure the team has good velocity and develops a good amount of output in the form of new and improved features. But too much focus on output brings risks of thinking about the individual features rather than the whole product, forgetting

about discovery, and of getting stuck nitpicking story point estimates and other proxies.

One key to striking the balance between output, outcomes, and impact is to focus at the right level of abstraction. I like to call this the epic-level, as it roughly corresponds to an epic in Jira, the popular issue tracking tool. I see two main benefits of putting emphasis on the word "epic". Firstly, it nudges the PM to think across the spectrum of output, outcomes and impact. Secondly, starting with the word "epic" and not always defaulting to the word "feature", can remind PMs that they should strive to escape the build trap[15] and that it is often more valuable to run some relevant experiments[16] and think in bets[17].



The best PMs avoid the trap of not paying attention to the customer. They manage the trick of prioritizing to maximize impact/output,

15 Escaping the Build Trap is a great book by Melissa Perri that explains this problem really well.
16 Here is an article on how PMs can turn experiments into bets, by Kate Leto.
17 Thinking in Bets is another great book, the title says it all, by Annie Duke.

while at the same time focusing most of their actual work on the customer outcomes[18]. Success is to innovate on behalf of your customers.

"Focus on customer outcomes. It is important both to make sure these outcomes link to business impact, and to slice features to make sure they don't require too much work output by the developers. Doing that allows you to say 'no' to some features. Then the most important thing is to focus on customer outcomes."

- Jens-Fabian Goetzmann, Head of Product at 8fit

In the next chapter, we will look at how to shift the focus of work for everybody to be more customer-centric by embracing user stories as a shared language.

18 Looking back at the example impact map from earlier, it also shows quite clearly that a big part of the work "happens in the middle". Skipping that middle part is a sure way to fail as a PM.

CHAPTER 2

Embrace user stories as a shared language

At this point, you have an impact goal and an idea for an epic, for simplicity's sake, let's assume the epic relates to building a feature. To figure out if that feature makes sense to do, and if so, what parts of it should be developed, and how, you need to collaborate with the stakeholders and different parts of your team.

Here we see that some PMs succeed in bridging the gaps between the perspectives of the different parts of their organization by working with user stories. They tend to do three things:

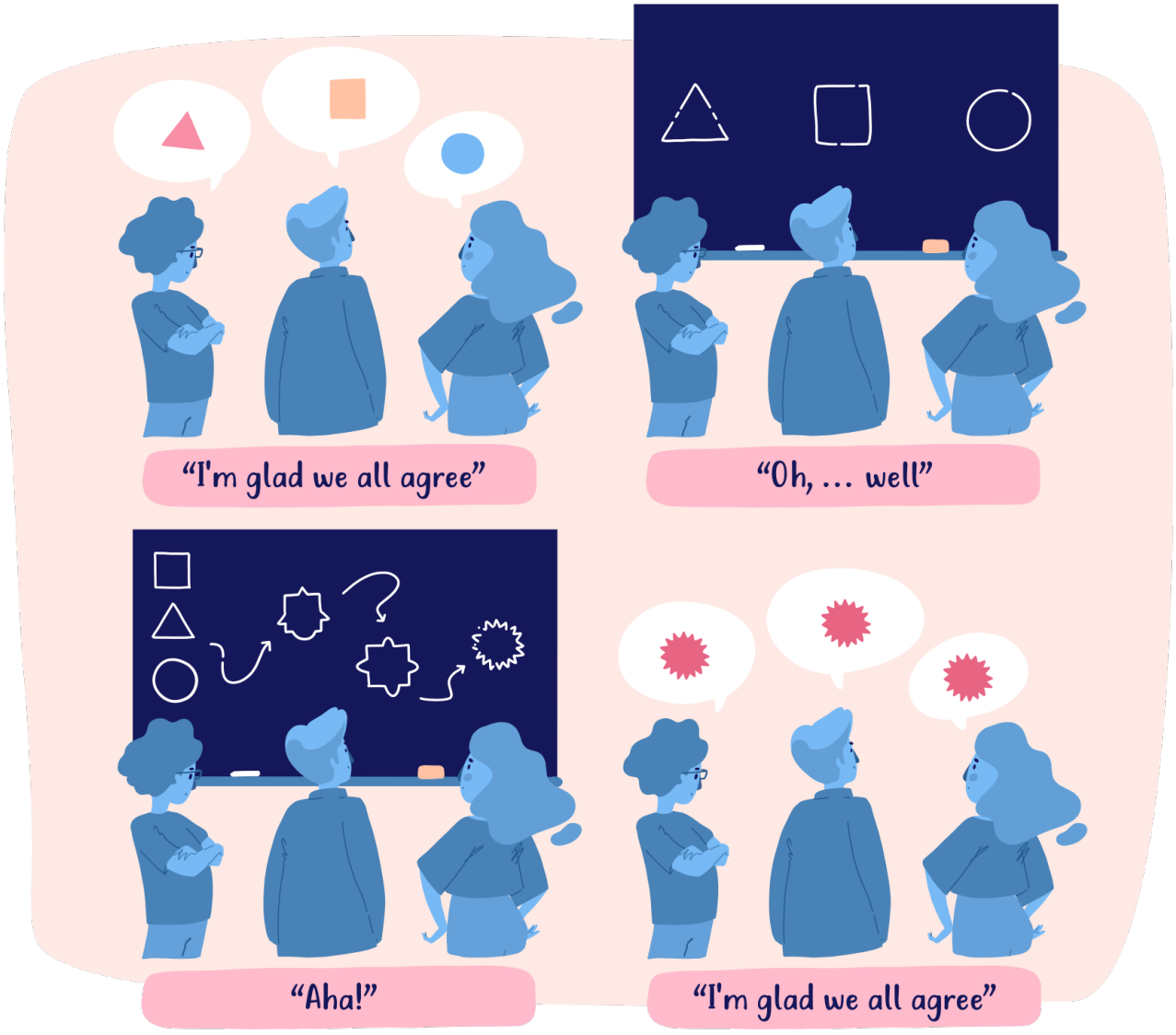
1. Express features as user stories
2. Do joint user story mapping sessions
3. Capture the conversation triggered by the user story

In this chapter, we'll look a bit more closely at these things.

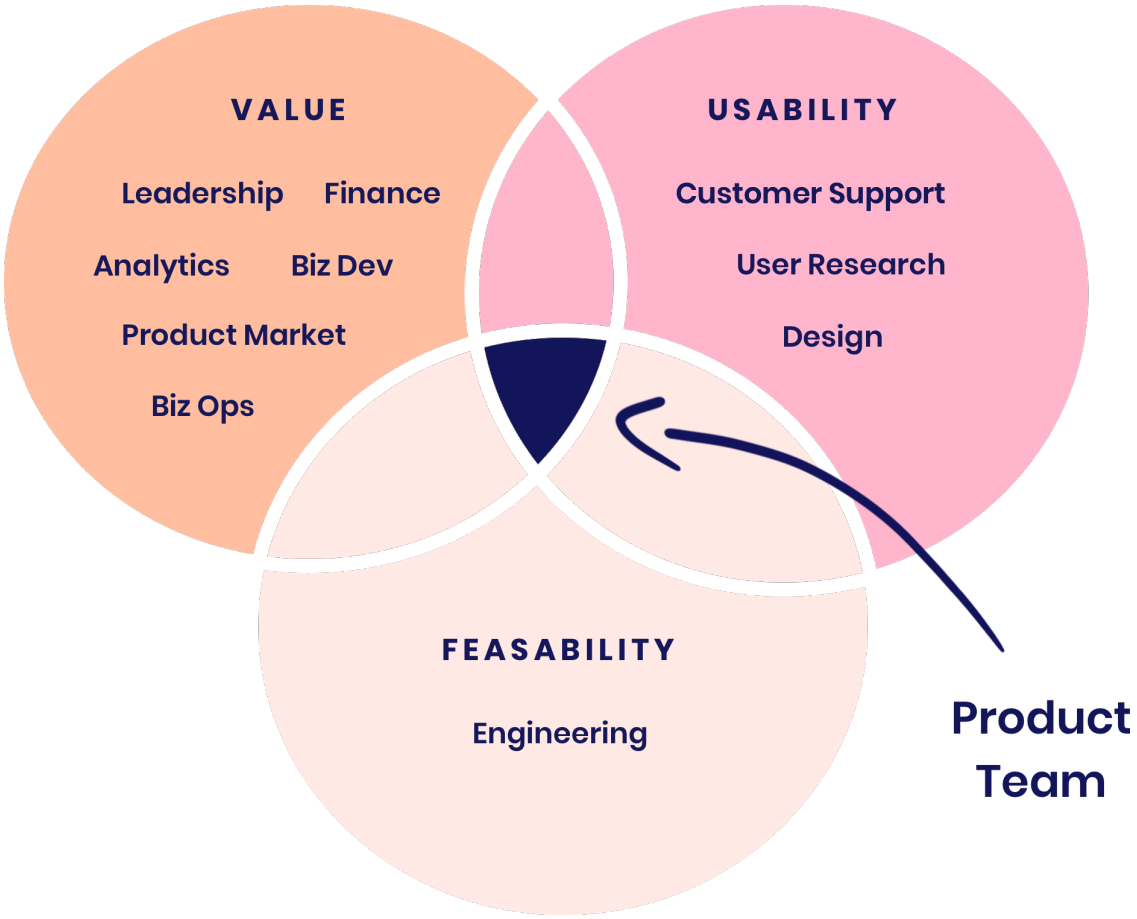
1. Express features as user stories

Because different stakeholders and parts of the team come with different perspectives, many PMs spend a lot of time translating across these different languages. But there is a better way of doing it.

Be aware of the different perspectives



Stakeholders and the different parts of your team think quite differently. A framework that describes their different perspectives is the Value-Usability-Feasibility framework:



These different groups not only think quite differently, they speak entirely different languages, using different concepts.

- Business people think in terms of e.g. customer requests, support tickets, and KPIs

- Designers think in terms of flows, wireframes, prototypes, and high-fi designs
- Developers think in terms of system changes needed, e.g. addition to a database, change to an algorithm or a new service or component.

Use the lingua franca[19] of product management

According to Wikipedia, "a user story is an informal, natural language description of one or more features of a software system. User stories are often written from the perspective of an end user or user of a system". There are several ways of writing user stories, but a common template looks something like:

> As a <role> I can <capability>, so that <benefit>

At first, most PMs find working with user stories a bit quirky. One PM likened expressing features as user stories to having to write all emails as haiku poems.

But after a while, most start to appreciate the almost magical way in which user stories manage to bridge conversations between the different parts of the team and the stakeholders.

- Business stakeholders, like customer support, can almost always give an answer to whether fulfilling a certain user story would solve a certain customer request.

19 A lingua franca is used to make communication possible between groups who lack a shared native language. Pretentious, moi?

- Designers can almost always arrange design flows along the lines of user stories (typically as part of a customer journey).
- Developers can almost always answer what needs to be developed to fulfill a certain user story (in combination with a set of acceptance criteria).

For the PM, the near-magical thing is that the formats of the answers from these different groups are completely different, but they can all relate to the concept of the user story, and whenever user stories are discussed, the focus is steered towards what is valuable for the customer.

Find your own pragmatic user story format

To many teams, the original user story format of "As a ... I can ... so that ..." feels too long and clunky. And using the original, full format everywhere is probably not the most effective, as the value of having a shorthand to refer to gets lost. But don't throw the baby out with the bathwater. In its entirety the original format brings value in that it reminds us to think about **context** and **intent**.

Most PMs use an abbreviated form, e.g. "Admin can grant user right to group", or even just "Grant right to group". This is much shorter and is thus practically usable as the short-hand name on tickets in the issue tracking tool (e.g. Jira), as it can be referred to in speech. And as long as the philosophy of the user story remains, and people understand why the user story is being built, it usually works well.

The risk with this pragmatic format is that it slides further and further away, to the point where it can no longer work as a shared language, for example "set groupRight" may be clear to the developers, but at that point the designers and business folks are probably no longer following.

Some teams don't have problems with this slippery slope, and some teams use the abbreviated form of the user story as the name of Jira ticket, but write the user story on the original, full format in the description field.

"One thing I've seen work really well to deliver on the spirit of user stories is to ask teams to connect specific user research to each user story--so that writing user stories actually encourages customer-centricity, rather than the formal format of the user story being a stand-in for customer centricity."

- Matt LeMay, Author of Product Management in Practice & Partner at Sudden Compass

Think "user stories", not "requirements"

Some teams use the term "PRD", which stands for "Product Requirements Document". If "Requirements" is just a word, that can of course be fine, but it is often problematic as the word has remnants of the Waterfall methodology, and those remnants tend to seep into the way work is done in practice. In Waterfall (as opposed to

Agile) development, the role of the PM is to a large extent that of a requirements gatherer, who hands down a list of requirements to the development team and is less involved in the process after that. There are two issues with this:

The first issue with thinking about "requirements" handed over from the PM to the team, is that it gives the team both less opportunity and less inclination to give input. The development team often knows better than the PM how long it would take to build certain functionality. Therefore, handing down a fixed set of user stories and acceptance criteria without any back-and-forth with the team before, risks going wrong in two ways. Either, the PM might "require" something that would provide value but be very time-consuming to do. Or, the PM might hold back on something that would provide some value, and would be very simple to do, because the PM wrongly believes it would be hard to do. Both scenarios would bring down the pace of value-creation by the team.

The second issue is that, in reality, a clean handover almost never happens. The initial requirements may have made perfect sense to the development team at the time of the handover. But new issues come up all the time, as reality often catches up with plans. Once it does, if the process implies a one-way handover, the team will be less likely to go back to the PM. As a result, there is a risk that the team spends its time building functionality that would not have been prioritized, had the PM known the actual amount of time required to build it. It's hard to swim up a waterfall.

2. Do joint user story mapping sessions

Individual user stories are great in that they allow people from all different parts of your organization to engage with them. However, PMs have to deal with a great number of potential user stories. We see that some PMs struggle with this complexity. The best PMs are able to engage in structured discussions that range across many user stories and thus manage to link feature outputs with customer outcomes. They often use some version of user story mapping[20].

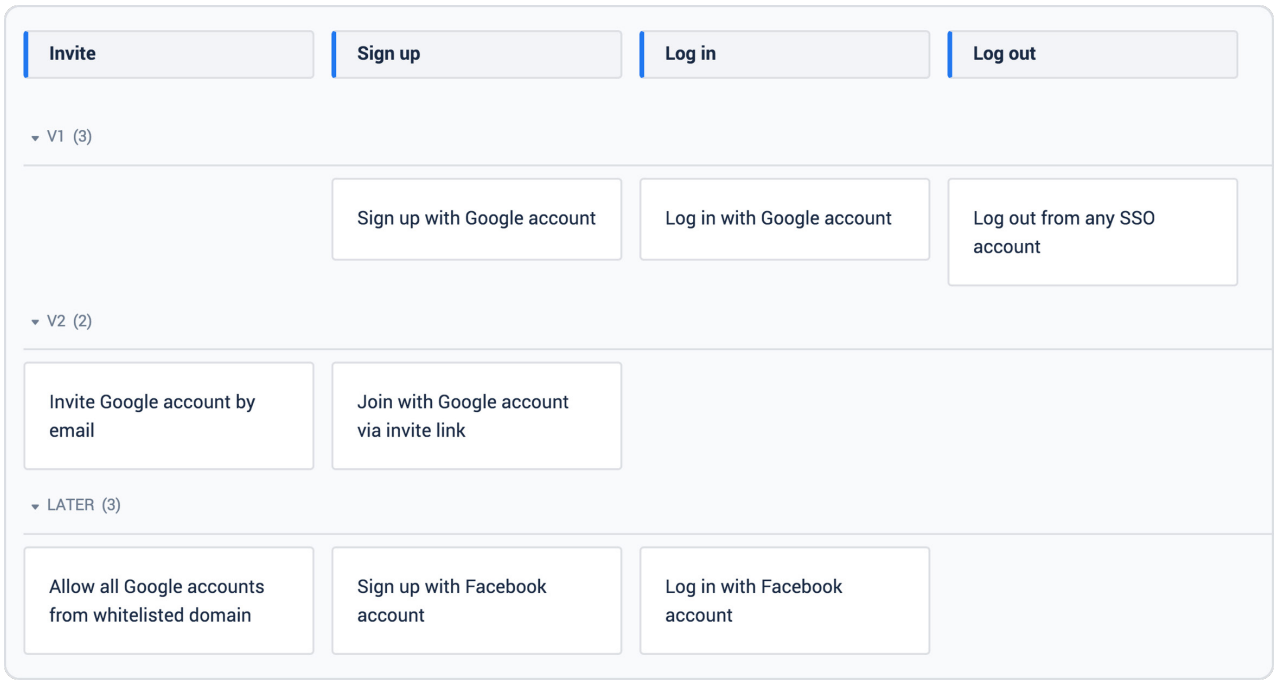
Combine the user stories into a map

The idea with user story mapping is to be able to have a conversation across several user stories. Generally, the approach is to visually place the user stories so that they tell a bigger story together, still from a user standpoint.

Specifically, a couple of new concepts are introduced (written in bold here, and then visualised in the image below). There is some variation in what exact terms are used, but for a simple user story map the concepts are as follows. The bigger story that the **user stories** tell together is referred to as an **epic** or a **feature**. This bigger story is divided into several **steps**, where several user stories could happen at each step. Together, the steps are referred to as the **narrative backbone** and form the columns of the user story map. Lines drawn horizontally represent different **releases**.

20 User Story Mapping is a method by Jeff Patton, that works really well.

To make it a bit clearer, let's look at a simple example:



Only map the steps you need to tell the story

The example user story map from the last section told the story of a single feature, i.e. that was its scope. User story maps can also be used at different scope levels.

Early on in the life of a product, one map can cover the whole product. For such user story maps, that cover a more complex bigger story, it is common to add structure to the narrative backbone, e.g. by grouping the steps by type of user.

Later on, that total map becomes rather large. Then it is perfectly fine to narrow the scope. You only need to extend the map with the steps

you need to tell a compelling story around the level you are currently working. The scope we see most is that of a single feature. For those maps, it is enough to include the steps taken within that feature, plus optionally a step before and after.

Tell the smallest story that results in the desired outcome for the customer

The genius with user story mapping is that by telling a coherent story from the customer's perspective for a release, you can use that as a test for whether the user stories included in the release will help the customer reach the desired outcome.

As you could see in the user story map example, the user stories are not only divided into columns depending on what step they help, they are also divided into rows based on what release they are to be included in.

This allows you to work with slicing the user stories into different releases.

The First release is sometimes referred to as the "Minimum Viable Product" or MVP (even though it in reality, typically is more of a minimum viable feature). The way to think here is to try to find the release with so few user stories that it just barely makes sense for the customer. This first release can be sliced very hard, including only a few user stories in total, and sometimes leaving some of the steps without a single user story.

For the following releases of the feature, you as the PM should continue to hold back on adding functionality, i.e., including more new user stories. You should always come back to the question of what customer outcome is improved by this next set of user stories. It only makes sense to build another release of a feature, if that release is one where the customer can achieve a better outcome.

Involve your team and stakeholders in user story mapping sessions

The reason for doing a joint user story mapping session is to be able to work together in prioritizing customer outcomes. By talking about the user stories along a narrative backbone (i.e. the steps of a user story map) seen from the perspective of the users, it becomes easier to be customer-centric.

By bringing in the different parts of the team and stakeholders, that customer-centricity can live throughout the development of the feature. Ideally, everybody that will be working with a feature should be in the session. At the very least, all different functions of your organization that will be involved should be represented (e.g., sales/support, design, development, QA).

Going into the session, you should have a hypothesis for the narrative backbone. It is also good to have an idea of some user stories to include, but it is good to keep them in your back pocket, at least initially, to let everybody else get going on the problem. And, as we described in the first chapter, the more context you bring with you,

the better (research, customer feedback, data, etc.).

Although there is almost always some iteration back-and-forth, the main steps of the session are to:

1. Agree on the narrative backbone
2. Come up with potential user stories
3. Slice the stories into the upcoming and following releases

The end-product of the session is a user story map for the feature, with quite a few user stories across the steps, divided into several releases, with agreement on at least the first release to go for. Beyond that there is hopefully quite a lot of notes from the discussion.

3. Capture the conversation triggered by the user story

The user stories are a great common denominator for all stakeholders and parts of the team. They can make sure that everyone thinks the same way at a higher level. But it is also important to keep everybody on the same page when going more into details.

A user story is a promise to have a conversation

The 3Cs framework[21] states that user stories have three critical aspects: Card, Conversation, and Confirmation:

Card refers to the user story, as user stories are sometimes written on index cards. The cards are not supposed to contain all the information about a feature, just enough text to identify it. It represents the feature and has notes on it that illustrate priority and cost. The user story is a promise to have a conversation.

Conversation refers to all communication among the stakeholders and the different parts of the team relating to the user story. Thoughts, opinions, and feelings are all part of the conversation. The conversation happens verbally in meetings, but also over email, as Slack messages, and as comments in different documents and tools. The conversation leads up to an agreed way to confirm whether the user story is fulfilled.

21 The 3Cs is a framework by Ron Jeffries, one of the creators of Extreme Programming (XP), and a signatory of the Agile Manifesto.

Confirmation refers to the set of acceptance tests that must be passed for the feature to be considered completed. These tests can have different formats, e.g. formal acceptance criteria or reference designs, and are used to remove any ambiguity as to whether the feature matches the customer requirements and has been implemented correctly.

Continue adding details to that conversation

User stories are great in that they can bring business people, developers and designers onto the same page. The goal of user stories is shared understanding, and the idea is that this shared understanding will come from an actual conversation with the people that matter, not from comprehensive documentation.

If you're using user stories and you don't write and draw to aid the conversation, you're doing it wrong. Ideally, the people that matter are in the whole conversation, and then the things written and drawn can serve as a reminder – in the same way that a vacation photo brings up more memories about the vacation than are shown on the actual photo[22].

But once the team has had the discussion in the same room, on that same metaphorical page, they tend to go back to their own area and do the work of actually detailing out how to implement the user story. Business people will, e.g. prepare the rollout plan, how to communicate, think about how to update the help center,

22 I think this is a great analogy from User Story Mapping by Jeff Patton.

etc. Developers will write e.g. pseudo code, data structures, and specifications. Designers will make e.g. flows, wireframes, sketches and prototypes.

As that work goes on, to avoid contradictions and mistakes, it becomes important to keep everybody on the same page, working from a single source of truth. In the next chapter, we will look more closely at how to make that as painless as possible.

CHAPTER 3

Use a single source-of-truth to create shared understanding

You talk about new features as user stories. But if each user story is a promise of a conversation, this means a lot of conversations. And on top of that, there are conversations happening above the user story level. These conversations, happening from ideation and early discovery all the way to deploy, will cover a lot of details, and the process is messy, with lots of people involved.

Many PMs struggle with this, and either spend lots of time updating across many different places, or simply don't capture details, and things get lost. The aggregated complexity of the PM job is so tough, that there just isn't any room for confusion about what document is the current one, or for that matter to run around and copy-paste and make updates across different documents.

During our research we found 5 practices common for great PMs who overcome this complexity and instead create a shared understanding among team and stakeholders

- Unless you share pizzas, write things down
- Capture most details in epic documents
- Make epic documents live across stages
- Use a template that can anticipate topics
- Get the structure you need with an outliner

Let's go through these practices, one by one.

1. Unless you share pizzas, write things down

As we've discussed in the last chapter, the majority of the feature refinement work falls between the Card (the starting point) and the Confirmation (the finish line), in the Conversation.

Trouble capturing the conversation

The Card is captured as a user story, and the Conformation is the story translated as acceptance criteria and/or reference designs. But the Conversation does not have any defined artifact like that. Instead, there are several communication channels that could capture it: whiteboards and presentations for live meetings, email, Slack, Jira (especially comments).

However, it is usually not possible to have everyone updated on everything, as that would require everyone to be on the same communication channel at all times, or read up on everything that has been said. People miss meetings, emails and messages. A developer might not understand what was needed. Sometimes missing a detail can lead to mistakes being made.

Whether or not this is a problem, comes down to pizza.

The two-pizza team

There are two cases where there is limited need to write things down from the Conversation.

One case is when everybody (both team and stakeholders) that will have input in implementing the feature in question is present during the whole Conversation. In other words, they don't need the updates and additional explanations, because they were present. To strive towards this case, Amazon has an ideal of the "two-pizza team". They try to keep the size of the group involved in any project co-located and small enough to be able to share and be fed by two pizzas[23]. We have talked to many PMs in small startups where everybody sits in the same room and they get by well, without writing much down. Conversely, we see that even smaller teams that don't sit together, have a need to to write things down in a more structured way.

23 For those of us not in the U.S. (in Europe, pizzas are typically made for one person), this is typically understood as around 6 team members, with a hard ceiling at 10. Never more than 10 members in a team.

Another case is when there is a self-contained issue, such as clear-cut bugs or obvious design improvements. In these cases a brief description can be, e.g. on how to reproduce. But teams should be aware that any issue might escalate into a lengthy discussion.

In all other cases, the team should make an effort to write things from the Conversation down.

Avoid disconnected conversations, write things down

In the ideal, two-pizza case, everybody that matters was in the room during the whole conversation, and can bring up memories of what was said by looking at notes and squiggles, like remembering a vacation by looking at a vacation photo. If that is not the case, the PM ends up trying to tell the story of a vacation to people who were not there, using just one photo, metaphorically.

Or put more plainly, the problem is that the conversation gets disconnected and spread across several meetings, communication channels, and platforms. This leads to difficulties in keeping everyone updated and on track and invariably to missed details, as making updates across all places just becomes too much. Disconnected conversations can make your job as a PM needlessly difficult.

The first step in putting the disconnected conversations back together is to write things down. Let's look next at how this can be done.

2. Capture most details in epic documents

To be able to piece together the different conversations that happen throughout the development process, most PMs write things down. A pattern for this that we commonly see, and that can be successful, is to write things in epic documents.

Write in one place, and if needed, refer to it

It is time-consuming and error-prone to make updates between tools and documents. When details about a feature (or an experiment or a bet) are first added to a new place, you need to put effort into either copy-pasting details or doing an entire re-write. And then if the information changes in one place, you then have a need to make updates across the different places.

- **Either you do update across.** If you do decide to keep updating information from and to both places after the transition, that is not only a lot of manual transcription work. But on top of that, work is needed to keep track of whether the different places are in sync. And with every update made, there is a risk that a mistake creeps in.

- **Or you don't.** If you don't, it is likely to cause loss of information during a handover, because the information was not moved. As a result, there could be inconsistency among documents. This means that the whole team might not be able to keep up with some important details, making part of the team misinformed.

Most PMs make updates some of the time. Working with multiple sources of truth is painful, as you as the PM are responsible for creating a shared understanding around features to build, and any inconsistencies in documents put you in a difficult position.

There are two ways of reducing the number of updates across tools and documents:

1. Reduce the number of places that contain details about the feature. This means reviewing the process of detailing out features, and removing some tools and documents, making sure they are no longer used.

2. For places that cannot be removed, refer back to another place, rather than adding redundant details there.

The best outcome is to have one place that is the single source of truth, which means not having to make repetitive updates of the same details beyond that place when changes occur.

The epic is the right level of scope for the document

When the scope of the document intended to concretely detail what to develop is too large, it tends to grow and become unwieldy and risk becoming obsolete. When the scope is too narrow, it risks not properly capturing the reasoning for developing the epic. Normally, a single user story is a too narrow scope, as it is often a combination of several user stories, an epic, that provides a customer outcome. That

is why the epic is the right level of scope for the main document to work with. It all depends, but if forced to generalize, we'd say that an epic should comprise of 3-12 user stories, and last for 1-3 sprints. If an epic risks lasting more than a quarter, it is probably better to restructure it.

The ideal is to have a single epic document that contains all the information about the feature/experiment/bet. Let's look at how close to that ideal it is possible to get.

Strive towards a single source of truth

Looking at how teams work with feature refinement, we find that they often capture details about the features they build in several tools and across several documents. Below are some practical suggestions for moving towards working with a single source of truth.

KEEP THE DETAILS OUT OF ROADMAPS

Most teams have a separate document for longer-term planning with features to develop or problem areas to focus on, i.e. a roadmap.

Some teams use special roadmap tools that can illustrate the roadmap akin to a Gantt chart[24]. Of these, many also let the roadmap contain more information about individual features, such as a more in-depth description of the feature, customer feedback that led to it, and different properties to use for prioritizing among features.

24 A Gantt chart is a type of bar chart that illustrates a project schedule.

However, it's not Agile to spend too much effort on making and following a plan (ref Agile Manifesto). We see great PMs coordinate with their stakeholders better by striking a balance and doing some light-weight and tentative planning.

It's not necessary to include all details about the features in the roadmap. The priority of a feature, what impact goal it relates to, and when to do it can be reflected in the roadmap, but it is better to put the rest of the why-what-how into the epic document. A straightforward approach is to have a document for each feature on the roadmap, and then link to the epic documents from the roadmap.

PRESENT DIRECTLY FROM THE EPIC DOCUMENT

We think making slide presentations tends to be a waste of time. We talk with PMs all the time who spend an inordinate amount of time creating and updating slide presentations aimed at explaining features to the team and stakeholders. Some PMs even create multiple versions of the same presentation for different audiences.

However, we also see some PMs that manage to show their epic documents directly to the people they are presenting to. This sometimes requires them to do a bit of negotiation, as stakeholders tend to prefer the slide format. And it requires that they keep their documents in order. But it is much more effective.

REFER TO DESIGNS

For features with UI components most teams have sketches, wireframes, prototypes, etc. The designers typically make these available to the team using their own tools. Sometimes they share their work by sending around different versions of design files. Each of these then become potential sources of truth, causing more complexity and confusion.

This is a part that most teams get roughly right. By referring from other documents to designs in e.g. Overflow, the product design communication platform, that they make sure are up to date, ambiguity about what the source of truth is, and number of places to update, is reduced.

REFER FROM TICKETS IN THE ISSUE TRACKING TOOL

Once the feature is detailed enough, tickets are added to an issue tracking tool (e.g. Jira). Often further details about the feature will be added in those tickets, to some extent a repetition of details from documents, and to some extent new ticket-specific details. This adds more potential sources of truth about the feature, and often the de facto source of truth moves to the different Jira issues when the transition is made.

This is problematic for PMs. On one hand, for stakeholders it is important that the information in the document that they look at is up to date. If the source of truth has been moved to Jira, then changes made in Jira will make the epic documents obsolete. The information

in the document might not be updated. However, the stakeholders could still believe that the document is up to date, and Jira tickets are often too granular for most stakeholders, which can then lead to misunderstandings.

We see three main ways of handling this problem: updating across, moving truth to Jira, and referring back to the document:

1. Some PMs put in the effort to update across to make sure the epic document and the Jira issues are in sync. If done well, this makes life easier for both stakeholders and developers but is time-consuming and error-prone.

2. Some PMs try to be explicit about the fact that the truth moves to Jira once tickets are created. One PM wrote "Obsolete. Source of truth moved to Jira." in red at the top of documents. This reduces the risk for misunderstandings and lets the developers work in their preferred flow, but there is work to move information to Jira, the flow is inaccessible to stakeholders (whom the PM must also get to agree to the flow), and for any thinking or change that needs to happen above the user story level, the format is awkward and cumbersome.

3. Some PMs make sure the epic document remains the source of truth. They write about the user stories in the epic document, and then instead of adding information to the Jira ticket, they refer back with a link to the relevant part of the epic document. This avoids the handover and duplication of information, accommodates late-stage changes better, and lets the

stakeholders see all information easily, but requires developers to navigate away from Jira. For this to work, the PM must get the developers to agree and actually check in the document so they can develop it as intended, challenge assumptions, and maybe suggest more feasible solutions. This means the document grows in length, but that can be handled by using an outliner, something we will come back to later.

Based on the PMs we have talked to, the ones that go for the third option tend to do best.

MERGE DOCUMENTS FROM ACROSS PHASES

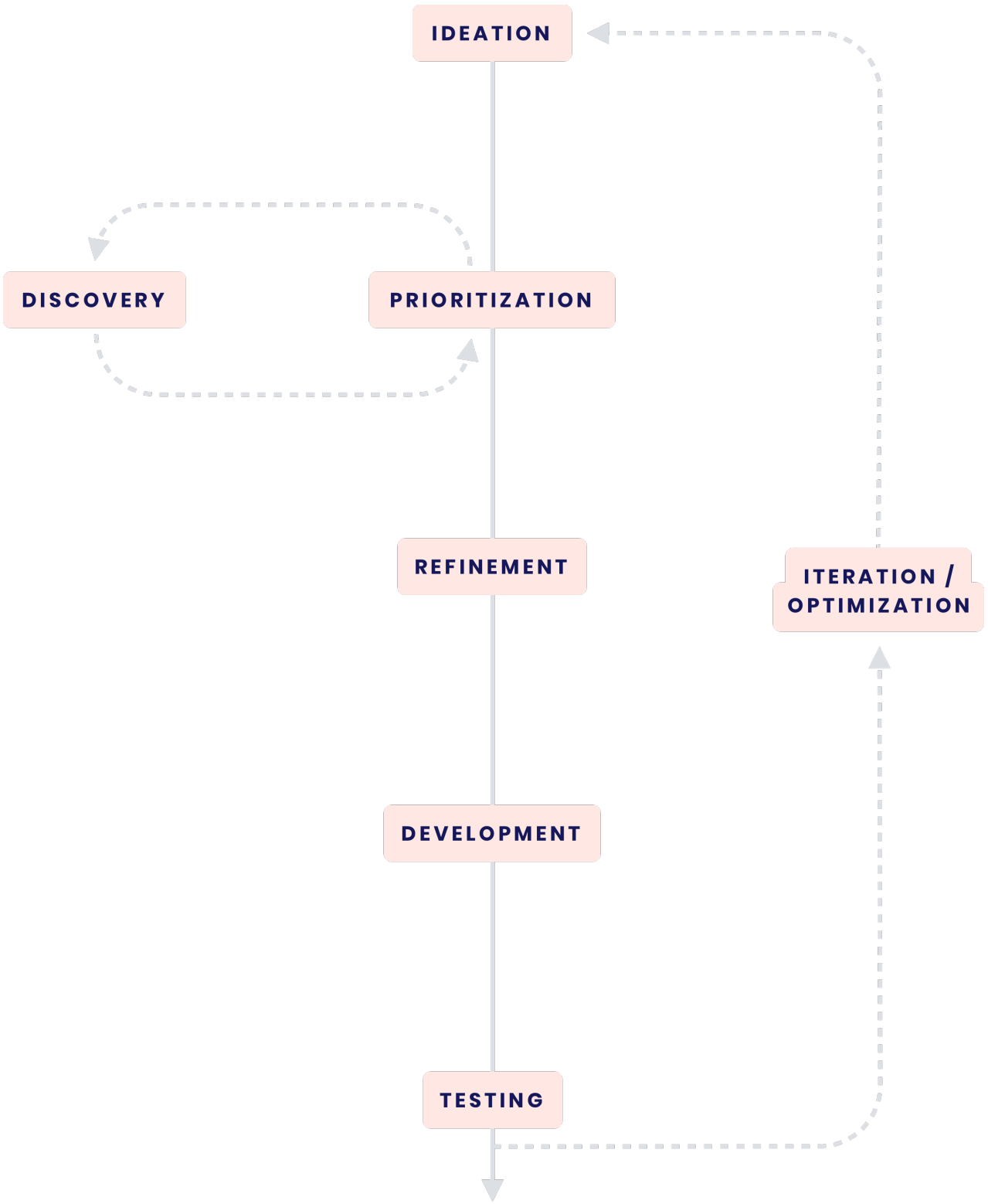
Teams often create several different types of documents throughout the process of developing the feature. Some examples are documents from the early ideation phase (support tickets and customer requests), from the discovery phase (customer interviews, technical investigations), or later phases (requirements documents and detailed specification documents).

Next, we will look at why it is best to put all details in those documents directly into a single epic document, and how to make that work.

3. Let epic documents evolve with its user stories across steps

A problem for teams is that with every step of the Archetypal Product Management Process (from ideation to prioritization to discovery to refinement to development to testing) there is a potential handover into a new document, with risk of extra work or information being lost and mistakes made.

The way to avoid this is to have the same epic document live across the entire process. User stories can help, but it is also helpful to look at how each of the different documents that teams create can fit into a single epic document.


IDEATION
DISCOVERY
PRIORITIZATION
REFINEMENT
ITERATION / OPTIMIZATION
DEVELOPMENT
TESTING

Benefit from user stories across clarification, scoping, and detailing of features

User stories are really useful. We have found that documents centered around user stories are more effective, both as they center naturally around what is valuable for the user, and as they provide a good structure to work with, across the different stages of feature development. The way to make a document truly centered on user stories is to let the document evolve with the user stories.

- In the early stages of writing the document, the focus will be on clarifying the problem and sketching a rough picture of what the solution might look like. Then it will be enough to write down potential user stories in the document.

- In the middle stages, the focus will be on agreeing on the scope of the feature. Then, the most important thing will be to prioritize which user stories to include, sorting them and moving some of them into a “later” subheading. To do this, especially for more complicated features, it can be a good idea to do a user story mapping session and then paste a photo from the whiteboard into the document.

- In the later stages, the focus will be to figure out more about how to complete the user stories in the first iteration. Then, it is time to add more details under each user story heading, e.g. acceptance criteria and use cases as well as designs, copy, flow, and technical micro-decisions. If done properly, this can save time in the transition to Jira.

Having the user stories in the document will give continuity as work progresses through the steps of the process. A common pattern we have seen among PMs to bring in user stories in their epic documents, is to have a section dedicated to user stories.

Work with the same epic document for all steps of the process

Across the different steps of the process, teams tend to create new documents and otherwise disperse information about new features. Let's look at a few of these situations, and how they can be handled.

CREATE THE EPIC DOCUMENT EARLY ON

Many teams use an "idea box" or something similar, where they gather internal ideas or feedback from customers. They do this as there can be a lot of ideas. But as soon as you start looking into an idea, try creating an epic document embryo.

There are several reasons for doing this. Many teams struggle with the distinction between the act of collecting ideas and actually refining them. Creating epic documents early on gives you a clear

way to separate these steps. If an idea comes up several times, each time can be added to the list of feedback in the epic document, as opposed to either having duplicate ideas, or starting to build out the idea in the idea box. Also, the act of articulating potential user stories for a feature as it is added to the epic document gives it more clarity.

WRITE DISCOVERY NOTES IN THE EPIC DOCUMENT

Spending time on discovery ensures that only features that make sense to do (i.e. that are valuable, usable, and feasible) go into development. The big risk with this is that discovery and delivery get too separated. It is surprisingly common that the actual feedback or data that led up to the feature being developed is not known to the developers implementing the feature.

The way to mitigate this is by writing the discovery notes in the epic document. That way, those who develop it can always just scroll up to see the problem statement as well as any recorded customer feedback or underlying data. (If you're thinking that these documents could get massive, don't worry. Last in this chapter we will describe how to mitigate this problem by using and outliner).

MAKE THE USER STORIES YOUR REQUIREMENTS

Some teams write a separate requirements document. All requirements are core to the feature and should be in the epic document. In fact it is often better to try to move away from the format of "requirements" and towards user stories with acceptance criteria.

At first glance, it might seem like a superficial change, but like we wrote earlier, the benefit of centering work around user stories is that they are better suited to being used as a shared language. While requirements risk becoming a separate list somewhere, the user stories and their acceptance criteria will always be at the center of the work for all parts of the team.

TECHNICAL DETAILS ARE ALSO WELCOME

It is common practice for tech teams to write separate documents for e.g. tech investigations and tech specifications for different features. By adding such details directly to the epic document instead, a handover and potential contradictions can be avoided.

For technical details that pertain to a particular user story, it is better to add them directly in the section for that user story, as they are then readily available to the developer picking the user story up in the issue tracker. The PMs that do this also tend to put technical details that relate to the entire feature in the epic document, in a separate section, continuing the pattern, and removing one degree of separation.

4. Use a template that can anticipate topics

Writing an epic document gives the team a central place to know what has been said and decided regarding the feature. To be useful, this document needs to cover the most important details, and so commonly runs to a couple of pages. With several PMs writing quite a few such documents, this amounts to a lot of information.

Some teams go all-in on writing things down but get overwhelmed with all the resulting text. Some teams balk at the prospect and don't write much down at all, with insights being lost and mistakes made. I believe that it is possible to get the best out of both worlds by working in a structured way, using a template as a starting point.

Let's break down why a template helps and what template specifically to use.

The magic of using a template

Every feature is different, and the details will vary. But looked at over time, epic documents tend to cover the same or similar topics. Therefore good PMs will typically find their writing style over time, and converge on a format to use as a starting point.

This works well with 1 or maybe 2 PMs in the organization. But, as the organization grows, so does the total number of people writing epic documents. Without any effort or thought going into this, the result is likely that these documents will look quite different.

Generally, for any process that is repeated many times, variance can be problematic. Specifically, if every PM has their individual style of writing documents, this will lead to two problems:

- It will be much harder for the person who will read these documents to find the information they want if all documents look different.
- If the PMs have not talked to each other to find a "best practice" for writing epic documents, it is likely that variance between how they write these documents means that some PMs do it less well. This likely means that feature refinement quality is suffering as a result.

The solution to problems with variance is to standardize. A good way to do this is to agree on a template for all PMs to use as a starting point when writing epic documents. To keep exibility and avoid limiting the PMs in their work, it is important that the template is just that, a starting point.

Using a template as a starting point for writing epic documents can yield three benefits:

It can ensure that the most important aspects of your features are covered. This, in turn, can increase the quality not only of the documents but the entire feature refinement process. Of course, this requires that your template actually covers the most important aspects.

- It can make it easier for the team and stakeholders to navigate within the document and find what they are looking for. If a consistent structure is used across documents, the documents will be readable and easy to use both for stakeholders and for the all development teams.
- It can reduce "writer's block". If you have ever dealt with it, you know that this can be quite a hurdle. Templates can help with this. Rather than starting from a blank page, you will have something to look at and start with, helping you place your thoughts into a structure. This will speed up the writing process.

Principles for a good template

By looking at documents from hundreds of the PMs we interviewed, we found some principles for how to write documents that help facilitate the conversation. The first three principles are related to best practices we discussed above, and the fourth relates to the user experience of the readers:

Epic level. The feature is the right level of scope for the document.

Focused on user stories. Epic documents centered around user stories can use the evolution of the user stories as an anchor throughout the process.

Both discovery and delivery. Epic documents should cover both discovery and delivery. The template needs to accommodate this,

with headings on both high-level items like what problem to solve and more detailed ones for e.g. tech details.

Structured for stakeholder readability. People will read the document in different ways. Executive stakeholders will read from the top and want a concise description of the problem, the solution, the measure of success, and the roll-out plan. The stakeholders that requested or are affected by the feature will also be interested in what user stories will be implemented. Specific functional stakeholders find it helpful to have a section of the document that speaks directly to them (e.g. QA, design, analytics, copywriting). The development team will go all the way into the details of each user story and down to further details.

Example epic document template

Based on the above principles and looking at lots of epic documents by great PMs, we developed what we believe is the best epic document template to start with:

Name of epic

Summary

Problem

Explanation for why we want to do this epic. Maybe some supporting feedback or data.

Solution

Overview of approach to solve the problem. Maybe a comparison of different solutions.

Measure of success

Description of how to evaluate whether the epic solved the problem.

Rollout plan

Actions needed to ensure the epic is successfully implemented

Stories

Solution as user stories. One bullet per story. Details like acceptance criteria as sub-bullets. To be linked to Jira once ready.

Later

User stories that will be sliced out, i.e. not included in the first iteration.

Details

Unsorted questions

Place to quickly add questions to resolve, to be moved to other section later.

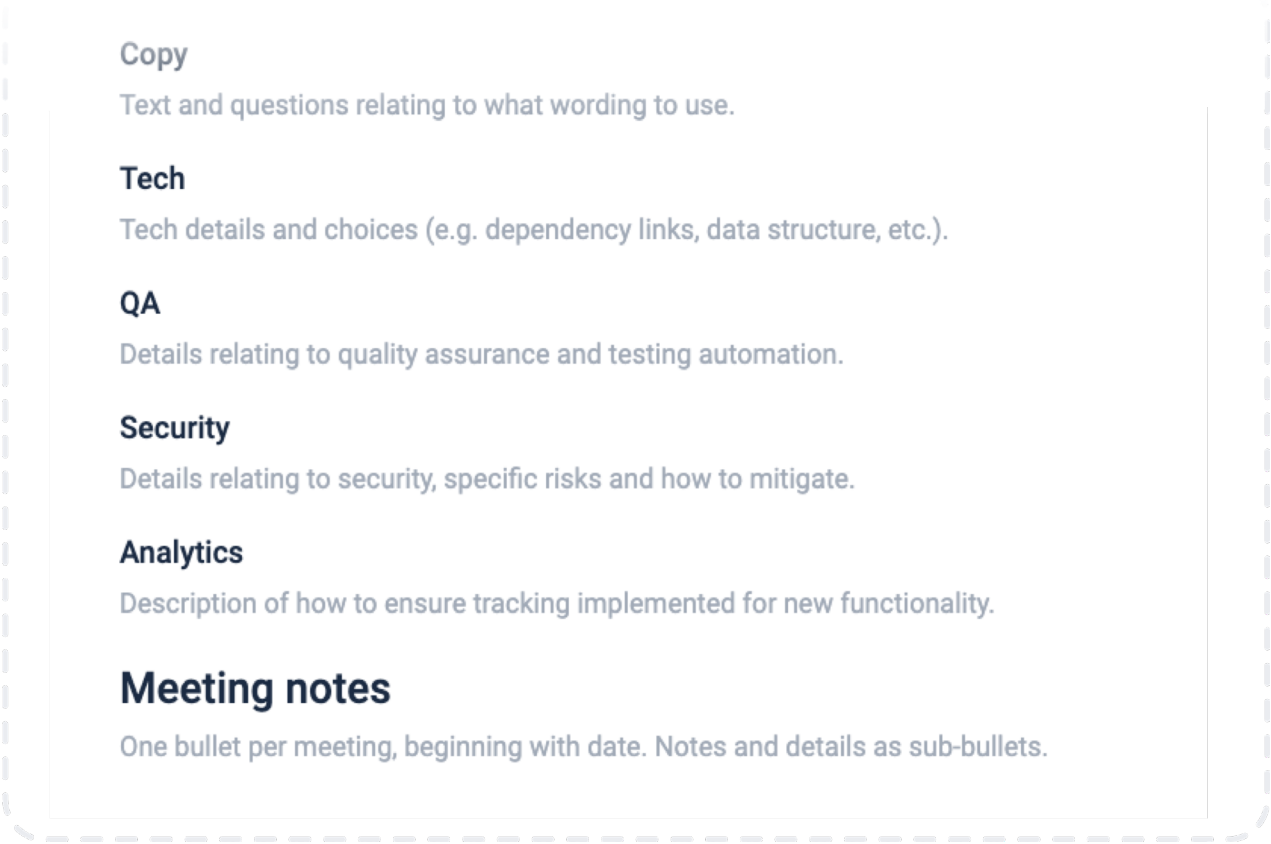
Scope

Questions relating to slicing, what to include in the first version of this epic.

Design

Text, links, images, and questions relating to user flows and designs.

Copy



But of course, the template is just a starting point. It should be used as a base and be open to changes. Depending on the feature, it can be adapted with judgment, with headings and sections added, changed, moved, or removed[25]. And working dynamically with the structure of a document is much easier using an outliner.

25 But whatever you do, make sure that your epic documents still have headings to answer the two questions "What problem are we trying to solve?", and "How will we measure success?".

"My team at Storytel has been using feature refinement templates for several months now. There are too many benefits to enumerate all, but giving a consistent structure to our projects is key.

Now everyone in the organization, regardless of position, knows what to expect when they follow a project link and knows where the information relevant to them will be in the structure.

We no longer struggle with keeping our projects sufficiently documented, we always start with a template and then we insist we do all our work – even Jira – from the document, refer back to the relevant part of the document, and then documentation is done automatically."

- David Božjak, Tech Manager at Storytel

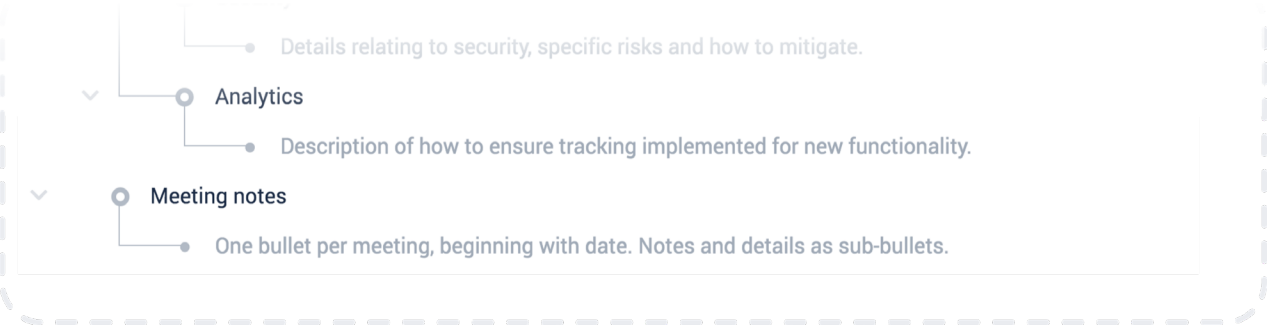
5. Get the structure you need with an outliner

An outliner is a type of word processor where the document is represented as a tree structure with bullets. Outliners are widely used by authors and screenplay writers to write synopsis for movies and books. Working with a document in an outliner gives the person writing more flexibility in engaging with the structure of the document, e.g. zooming into, collapsing, or moving around parts of it.

Below is an example of what a document can look like in an outliner. It is the same template from the last section, but looked at with an outliner:

Name of epic

- Summary
  - Problem
    - Explanation for why we want to do this epic. Maybe some supporting feedback or data.
  - Solution
    - Overview of approach to solve the problem. Maybe a comparison of different solutions.
  - Measure of success
    - Description of how to evaluate whether the epic solved the problem.
  - Rollout plan
    - Actions needed to ensure the epic is successfully implemented
- Stories
  - Solution as user stories. One bullet per story. Details like acceptance criteria as sub-bullets. To be linked to Jira once ready.
  - Later
    - User stories that will be sliced out, i.e. not included in the first iteration.
- Details
  - Unsorted questions
    - Place to quickly add questions to resolve, to be moved to other section later.
  - Scope
    - Questions relating to slicing, what to include in the first version of this epic.
  - Design
    - Text, links, images, and questions relating to user flows and designs.
  - Copy
    - Text and questions relating to what wording to use.
  - Tech
    - Tech details and choices (e.g. dependency links, data structure, etc.).
  - QA
    - Details relating to quality assurance and testing automation.
  - Security
    - Details relating to security, specific risks and how to mitigate.
  - Analytics



Working with a template helps make sure the document contains all the relevant information. However, it creates a problem in that the document risks growing in size and becoming unwieldy. That problem is to a large extent mitigated by using an outliner, as you can collapse it, and let readers see only the stuff that is relevant for them.

In combination with a template, an outliner brings four benefits:

- First, by collapsing the document, a reader can navigate among the headings of the template faster, even if lots of details have been added to the document. A reader that is new to a document does not have to be overwhelmed by all of the content, but can dive directly into the parts relevant to them.
- Second, working from an overview of the headings in the document, it becomes simpler to add things to the right place, as there is no longer any need to scroll around looking for a specific part of the document. This makes it easier to adhere to the structure of the document.

- Third, the structure of a template should always just be seen as a starting point, not a rigid structure to be conned by. Being able to see the structure of the document, and being able to make changes by drag-drop or keyboard shortcuts, using an outliner makes it much more straightforward to change the structure, revising the hierarchy of headings or adding new sections.

- And last but not least, once you become comfortable with working in an outliner, you will start using keyboard shortcuts more and more and find that it also allows you to, for example, split, merge, and move around paragraphs much faster. You may not become an author overnight, but it will make you a more effective writer. And more well-written and engaging epic documents are more likely to be read, which is of course useful if the goal is to create a shared understanding.

“I love using an outliner. I use it for everything I ever write nowadays and I think the reason it’s so powerful is that it kinda mimics how both conversation and thoughts generally work.

For feature documents specifically the outlining functionality is the one thing that makes it possible at all to keep these documents single sources of truth with all the complexity and details that are needed at the same time as making it possible for more remote stakeholders to quickly get an overview.“

- Mikael Holmquist, Product Manager at Storytel

“I use an outliner to write structured feature documents to give my team and external stakeholders a high level overview with the ability to take a deeper dive into the features we are building. Not only does it keep initiatives in context, but also it allows me to “double-click” and present more detail about the design or technical execution all in one place.

Using an outliner streamlines my ability to organize, plan, and present into one document, which saves me time, reduces errors, and keeps everyone on the same page.”

- Sun Kim, Senior Product Manager, Kabbage

CHAPTER 4

Expose micro-decisions through questions

You work your way through the complexities of the job by prioritizing impact while focusing on customer outcomes, talking in the language of user stories, and capturing it in epic documents as the single source of truth. So far so good. But to really get through all of the aggregated complexity, there are a lot of details that you need to get people to agree on. These details are what we refer to as micro-decisions.

Here we have seen PMs come out on top by doing five things:

1. Recognize the many micro-decisions
2. Frame decisions as explicit questions with decisions as answers
3. Practice the art of asking the right questions
4. Facilitate structured discussions for the difficult decisions
5. Make the decision-making process transparent within the doc

1. Recognize the many micro-decisions

Most PMs don't think about it this way, but a core part of their job is facilitating decision making. For every feature they build, they need to make sure lots and lots of decisions are made. Some are obvious decisions, such as when to build the feature and what functionality to include in the first iteration. But if you think about it, almost every aspect of a feature can be boiled down to a decision that needs to be made.

To emphasize this multitude of decisions, we like to talk about micro-decisions. Micro-decisions can be of all kinds. Their topics range from scope to tech to design to copy. For example, when choosing a tech library or determining what data structure to use, you're making a micro-decision. Setting the sequence of a user flow and selecting what type of component to use (e.g. modal vs. popover) are also micro-decisions.

Facilitating all these micro-decisions effectively is a real challenge for many PMs. Using the epic document as the single source of truth for all the details give PMs some control. But as long as the team is working on a feature, the epic document will to some extent be in flux. It will be discussed during a number of meetings.

To better stay on top of not only of what has been decided, but also on what has been discussed so far, and what is left to decide, you can distinguish between three types of comments in these meetings:

1. Comments directly relating to one aspect of the feature, e.g. “What tracking should we add for this feature?”

2. Comments relating to the discussion on one of the micro-decisions, e.g. “I can’t remember, what were the reasons for that decision?”

3. Comments relating to the overall status of the feature, e.g. “What is left to decide for this feature?”

However, many PMs struggle – keeping up all this on a daily basis is difficult. Specifically, two parts of how PMs handle these challenges warrant further attention:

1. How they frame micro-decisions to make them easy to discuss and come to a joint conclusion on.

2. Where and how they note down and follow up on micro-decisions to make it easy to see what has been decided, why, and what is left to decide.

In our interviews, we saw many different approaches to this. We also noted a wide variance in how well PMs thought their approaches were working. The way PMs work with micro-decisions can have a great impact on the extent to which they manage to create shared understanding. Yes, the devil is in the details.

2. Frame decisions as explicit questions with decisions as answers

It matters how PMs try to capture the decisions that are being made. It almost always makes sense to take notes on decisions relating to the feature. In theory, there are two exceptions, either if everything about the feature is entirely obvious, or if everybody giving input to or working with the feature will be present in all meetings where the feature is discussed. But since these rarely occur, PMs do best to take notes on decisions.

When writing down information about a decision, the way it is written down can affect how easy it will be to collaborate around. There are three main ways of framing decisions, with increasing degrees of explicitness:

1. IMPLICIT DECISION IN TEXT

One common way decisions can be captured in a document is by being referred to implicitly.

Let's clarify by looking at the example of choosing OAuth as the standard for user authorization. Implicitly capturing such a decision could mean part of a technical description of the feature contains text along the lines of:

> *"..then OAuth access tokens will be sent.."*

The benefit of this approach is that it is lightweight and requires little upfront time or effort. But the decision lacks visibility. This increases the risk of the decision not being noticed by people that may have a valid objection. It also increases the risk that different parts of the document imply contradictory decisions .

2. EXPLICIT DECISION

Another approach for capturing decisions is by simply stating the decision in plain text. Using the same example of OAuth, this may look something like:

> *"Users will be authorized using OAuth."*

This approach requires some effort but solves the problem of decision visibility. However, a person reading this does not get any sense of the considerations that preceded the decision or if any alternatives were considered.

3. EXPLICIT QUESTION WITH DECISION AS ANSWER

The third approach is to explicitly state the question that the decision tries to answer, and then write the decision as an answer to that question.

For OAuth, this may look like:

> *"What standard to use for user authorization? OAuth."*

This approach requires a bit more effort upfront, as it forces thinking on what underlying question the decision addresses. Writing both

the question and the decision also typically makes the document a bit longer, as opposed to just writing the decision.

But this approach also brings a number of benefits:

- Writing the question also means thinking about the problem being solved.
- Asking a question opens up the thinking of people who read the document, making them think also of alternative answers.
- Evaluating different alternatives typically yields better decisions.
- Writing the rationale down makes it easier to come back to the decisions.

3. Practice the art of asking the right questions

Many decisions benefit from being formulated as answers to a question. And the effort of doing so is often worthwhile. When formulating explicit questions for your micro-decisions, there are two things to think about:

Use judgement when evaluating the number of micro-decisions to write out explicitly as both question and decision

Too many questions and you won't see the forest for all the trees. Too few questions and it is likely that you are not exposing your thinking enough to your team/stakeholders to enable them to give you feedback. For example, questions where the answer would be a point of fact rather than a decision do not need to be written out. The litmus test for whether to write out a micro- decision as an explicit question is:

"Is this an obvious choice, or is it possible that any stakeholder/team member would object to this?"

So, how many questions should there be for a feature? – "How long is a piece of string?" – no seriously, it of course depends, but if forced to generalize, based on what we have seen, 10-20 questions are probably the right amount for a feature that takes one two-week sprint to complete.

Aim to formulate unbiased questions with options that can be expressed concisely

To simplify, there are three types of questions: Yes/no, multi-option, and open. Some decisions are better stated as yes/no questions, but often it is meaningful to rephrase them as multi-option questions, as that opens up the thinking to alternative answers. Don't use questions that are too open, as these are harder to give a clear answer to, and thus to make a decision on. Instead, break them down to questions that can have concise answers. Moreover, formulate your questions in an unbiased way, even when there is a strong hypothesis, making a preliminary decision is a better way to reveal the hypothesis.

4. Facilitate structured discussions for the difficult decisions

Most of the micro-decisions captured as questions can be decided rather quickly. But for most features, there are a few questions that risk triggering long-winded discussions where people talk past each other, both wasting time and causing frustration.

PMs have a lot to gain by sensing whenever one of these thorny discussions are coming up, and then stepping in as a facilitator. With some structured facilitation, most discussions can be fairly quickly resolved.

One framework for such structured facilitation is the Six Thinking

Hats. As a pedagogical device, it represents different kinds of thinking with six hats[26]:

> **Blue hat – The Big Picture** – Thinking about thinking, where are we on the agenda, what questions should we discuss?
>
> **White hat – Facts & Information** – Identifying and presenting information that is needed, what do we need to know to have an informed discussion?
>
> **Green hat – Alternatives and creativity** – Coming up with ideas for answers to the questions posed, what if we do it like this?
>
> **Black hat – Critical Judgement** – Highlighting problems and risks, coming up with cons for different options
>
> **Yellow hat – Positive Judgement** – Emphasizing upsides and benefits, the good things, coming up with pros for different options
>
> **Red hat – Feelings & Intuition** – Gut instinct on what to do, either in the early phase of the discussion or later on taking pros and cons into account, what option do you think is the best?

You can facilitate tricky discussions by adding direction regarding what hat the participants should put on, for example:

- If one alternative seems too good to be true, you can say "let's put on the black hat, what are the problems or risks?"

26 From the book Six Thinking Hats by Edward de Bono.

- If it seems the team is stuck with two bad alternatives, you can say “let’s put on the green hat, are there any other alternatives here?”
- If it is not clear what it is the team is discussing, you can say “let’s put on the blue hat, what is the question we are trying to answer here?”
- And so on...

Adding structure, balance, and direction to a discussion can be incredibly powerful, and allow the team to make better decisions, in less time, while still having a high degree of involvement.

“I find it incredibly powerful to have the mindset of evaluating alternatives. So many times I find a single solution to a problem described in documents, without even considering that another solution could be possible. Often there are other solutions, sometimes they are better. By making the evaluation of alternatives explicit, we can make much better decisions.”

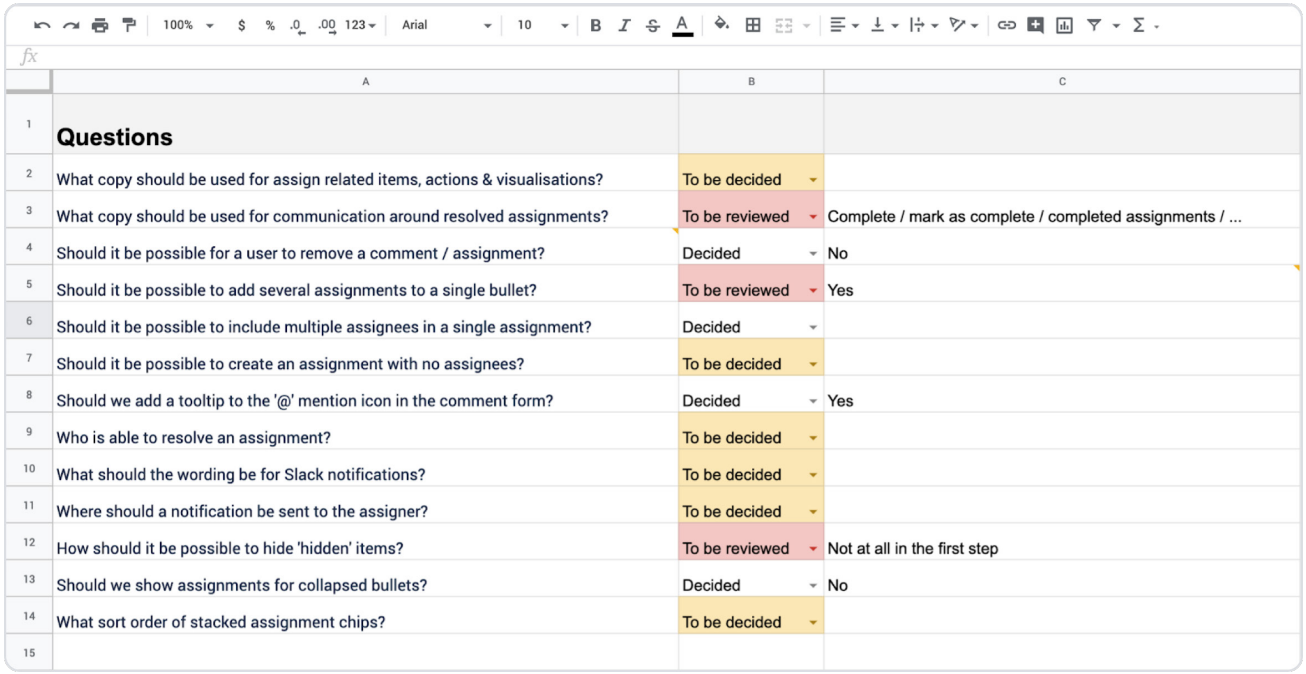
- Staffan Eketorp, Software Engineering Manager at Apple

5. Make the decision-making process transparent within the document

PMs who are diligent about capturing micro-decisions and framing them properly run into the next problem: Now there are a lot of decisions to facilitate and follow up on.

In our interviews with PMs, we found four different ways of dealing with this challenge, each with its pros and cons:

1. SEPARATE SPREADSHEET

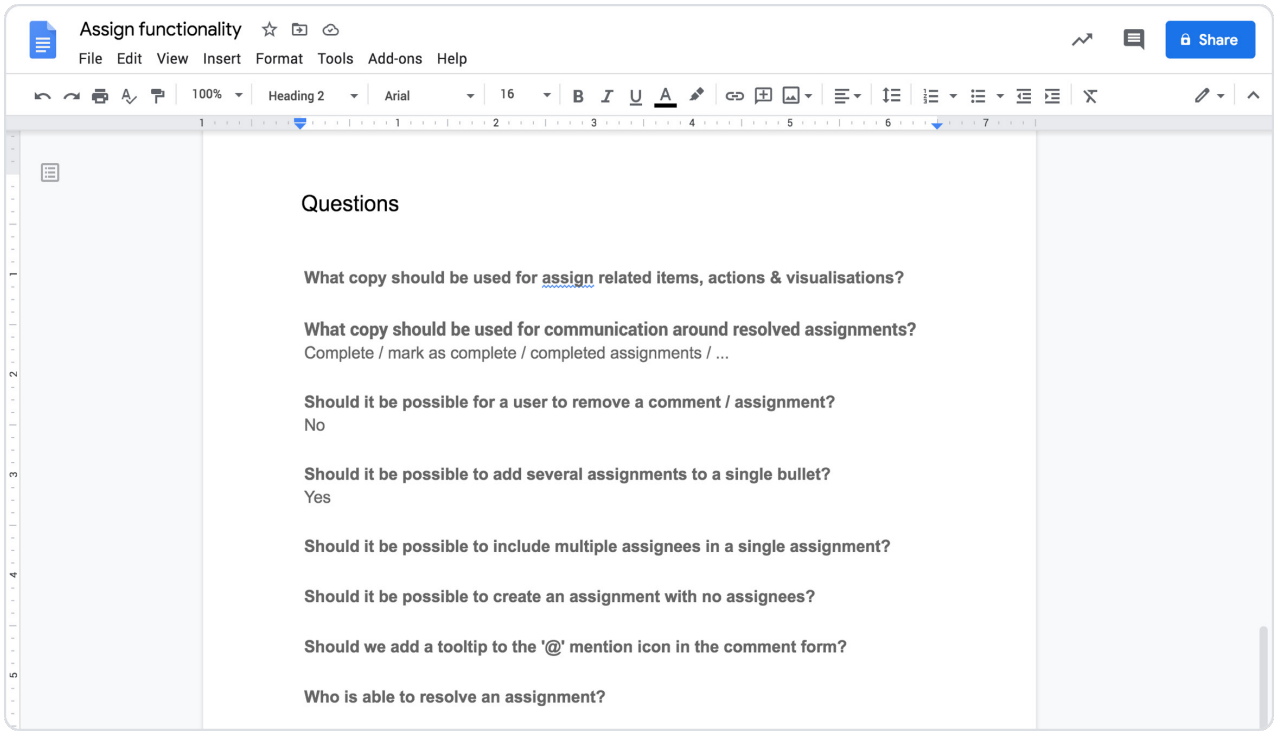


	A	B	C
1	**Questions**		
2	What copy should be used for assign related items, actions & visualisations?	To be decided	
3	What copy should be used for communication around resolved assignments?	To be reviewed	Complete / mark as complete / completed assignments / ...
4	Should it be possible for a user to remove a comment / assignment?	Decided	No
5	Should it be possible to add several assignments to a single bullet?	To be reviewed	Yes
6	Should it be possible to include multiple assignees in a single assignment?	Decided	
7	Should it be possible to create an assignment with no assignees?	To be decided	
8	Should we add a tooltip to the '@' mention icon in the comment form?	Decided	Yes
9	Who is able to resolve an assignment?	To be decided	
10	What should the wording be for Slack notifications?	To be decided	
11	Where should a notification be sent to the assigner?	To be decided	
12	How should it be possible to hide 'hidden' items?	To be reviewed	Not at all in the first step
13	Should we show assignments for collapsed bullets?	Decided	No
14	What sort order of stacked assignment chips?	To be decided	
15			

Some PMs keep a separate spreadsheet where they track all the decisions to be made relating to a certain feature (as in the image below).

This approach makes it natural for PMs to formulate decisions as explicit questions as well as show decision status and other comments. However, it separates the questions from their context into a separate document, which means the PM needs to work with two sources of truth, one for the process and one for the content. This can be cumbersome and risks leading to mistakes.

2. SEPARATE SECTION

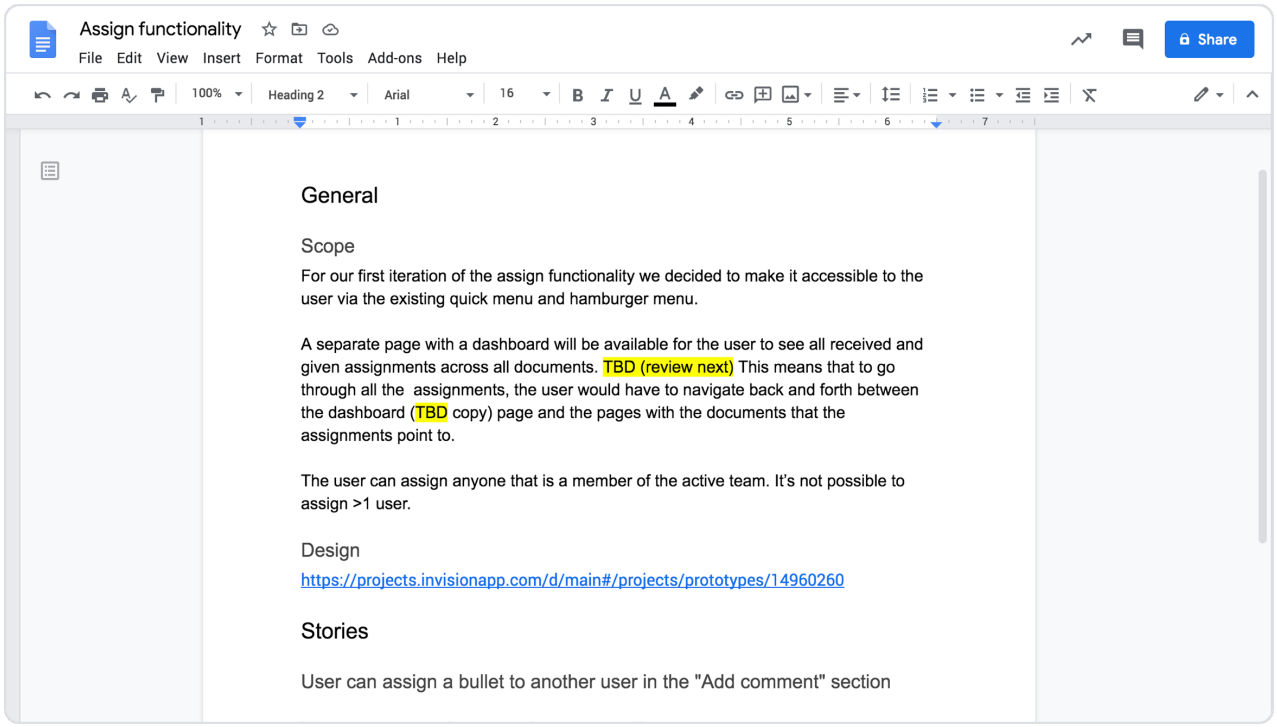


Some PMs have a separate section in their epic document where they write down questions that need a decision (see image below).

This approach also makes it natural for PMs to formulate decisions as explicit questions, but decision status is typically not as clear. Also, even though the decisions are now in the same document, they are

still somewhat separated from context, which typically leads to a lot of scrolling.

3. DOCUMENT HIGHLIGHTS OR COMMENTS

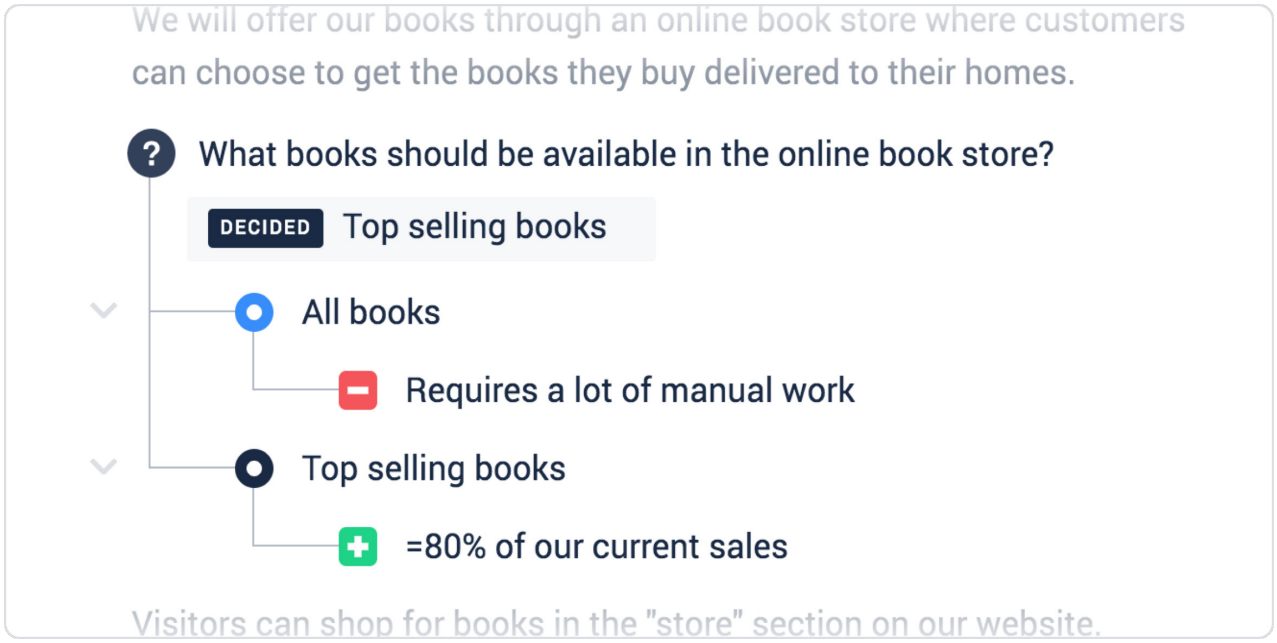


Some PMs make highlights and/or comments in the sections of the document that the decisions pertain to. A common pattern is to highlight parts of the document to indicate that a decision is needed, another is to call out that a decision is needed using document commenting functionality (the image below shows what highlights might look like).

Unlike the other two approaches, the decisions are not separated from the context. However, as the decisions are not stated as questions, the status of the decision is not always clear, and comments can be somewhat disorganized. Also, once the decision is

made, highlights and comments are typically removed, which makes it difficult to know if there was any discussion, and if so what was said.

4. WITHIN RELEVANT SECTIONS



Some PMs write decisions as explicit questions in the sections of the document that the decisions pertain to. This is typically done by writing the question on a new line. Alternatives considered and other details on the discussion can be added and indented below (see animation below). This approach ties together the best parts of the previous approaches, and we found that PMs that use this approach managed to combine working within a context with clarity regarding decisions being made.

We see that option 4 typically works best (question within the relevant section). If a question has come up and it is not yet clear

where in the document a question should go, option 2 can make sense (separate section in the document with open questions). If the underlying question is not yet clear or to indicate parts of the document that need more work, option 3 can make sense (highlights and comments). We see some meticulous PMs make option 1 work (separate spreadsheet), but they often both save time and make their decision making more transparent by moving to option 4.

What it can look like in practice

We have gone through four approaches to working with feature documents. I hope you have found them helpful. During the many revisions of this book (far too many, I have to admit), one consistent feedback that I received was to add more examples. Therefore, in this final chapter, I have tried to create one example epic document and briefly point out how it illustrates the approaches.

About the example document

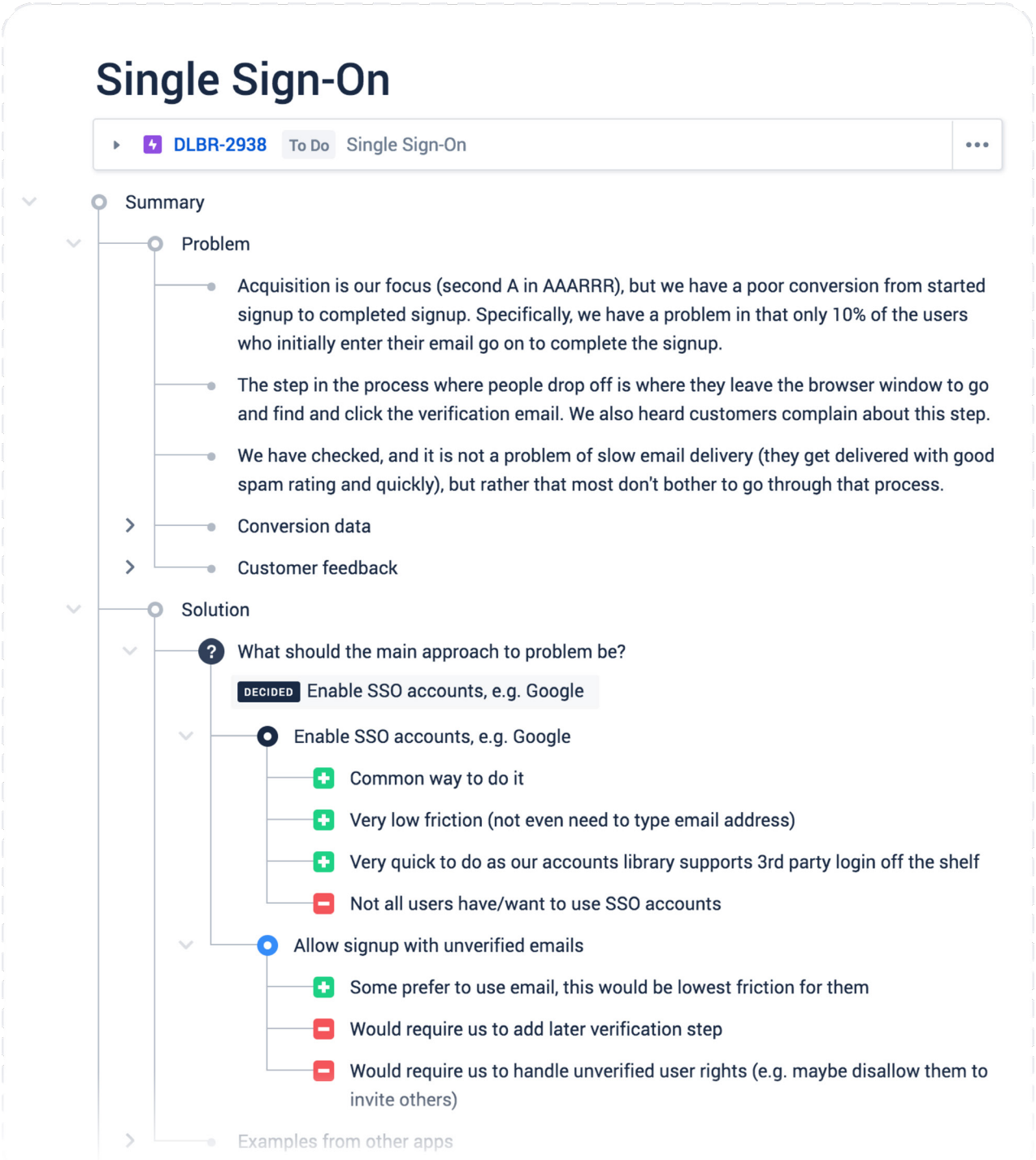
Before going onto the example itself, I'd like to first add some caveats and then position the example a bit. The approaches described in the book are more high-level guidelines to be applied with judgement than a detailed set of rules to follow to the letter. Also, I have seen many different takes work well, applying the approaches slightly differently. Because of this, I was a bit hesitant to put my foot down and say "this is what a good epic document should look like". Nonetheless, because you guys asked for it, below is my take on what a good epic document could look like. Please take it for what it is, an example, and be mindful that a lot of the choices depend on where you are as a company, what stage, what priorities, what constraints, what type of feature, etc.

To make it as relatable as possible I tried to pick one of the epics I have seen the most times - the adding of Google signup/login. Maybe you have done it (and recognize the thinking) or maybe you are about to do it (and get some clarity from the example). It is an amalgam of a lot of different documents doing more or less the same thing.

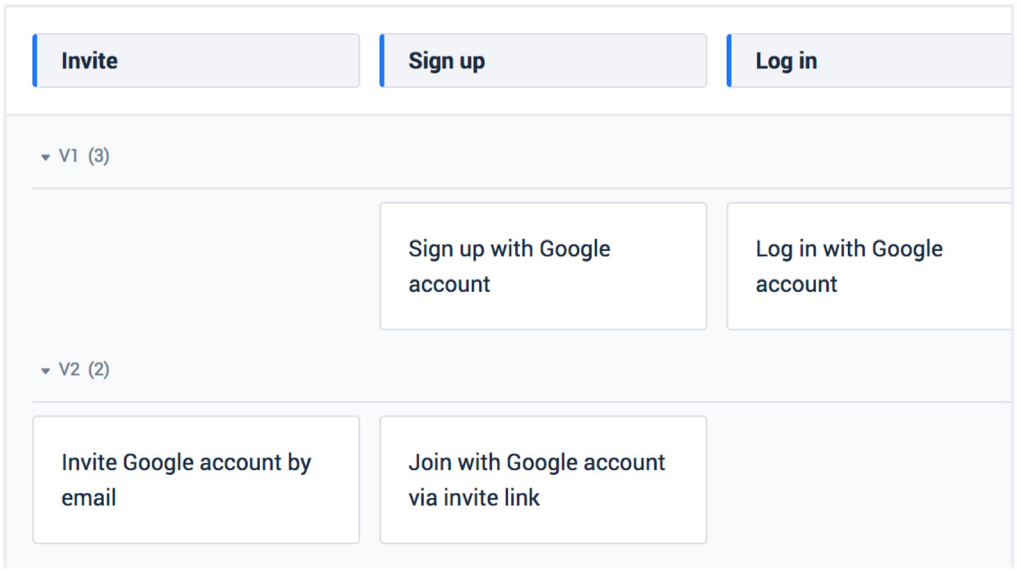
To avoid overloading you with too many details at the same time, I have collapsed part of the document, for example I think you can imagine what might be under the headings “conversion data”, “customer feedback”, “examples from other tools”. I also collapsed all user stories except one, to illustrate the point. In the details section, I collapsed most sections but kept “analytics” expanded, as I think some details there relate to the first chapter about driving towards impact. In general, when using documents to communicate with different stakeholders, it is good to expand the parts relevant to them, and collapse most of the rest. With that in mind, the expand-collapse state of the document as shown below can be said to be roughly tailored to a discussion with an executive stakeholder and the business analyst that will implement the tracking and setup the relevant analytics dashboards.

Without further ado, here is the document. In the section after, I will reason about how the four different approaches can be seen throughout the document.

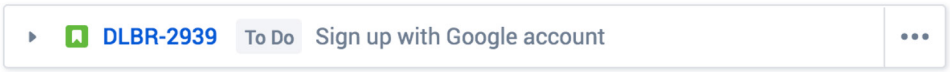
Example epic document



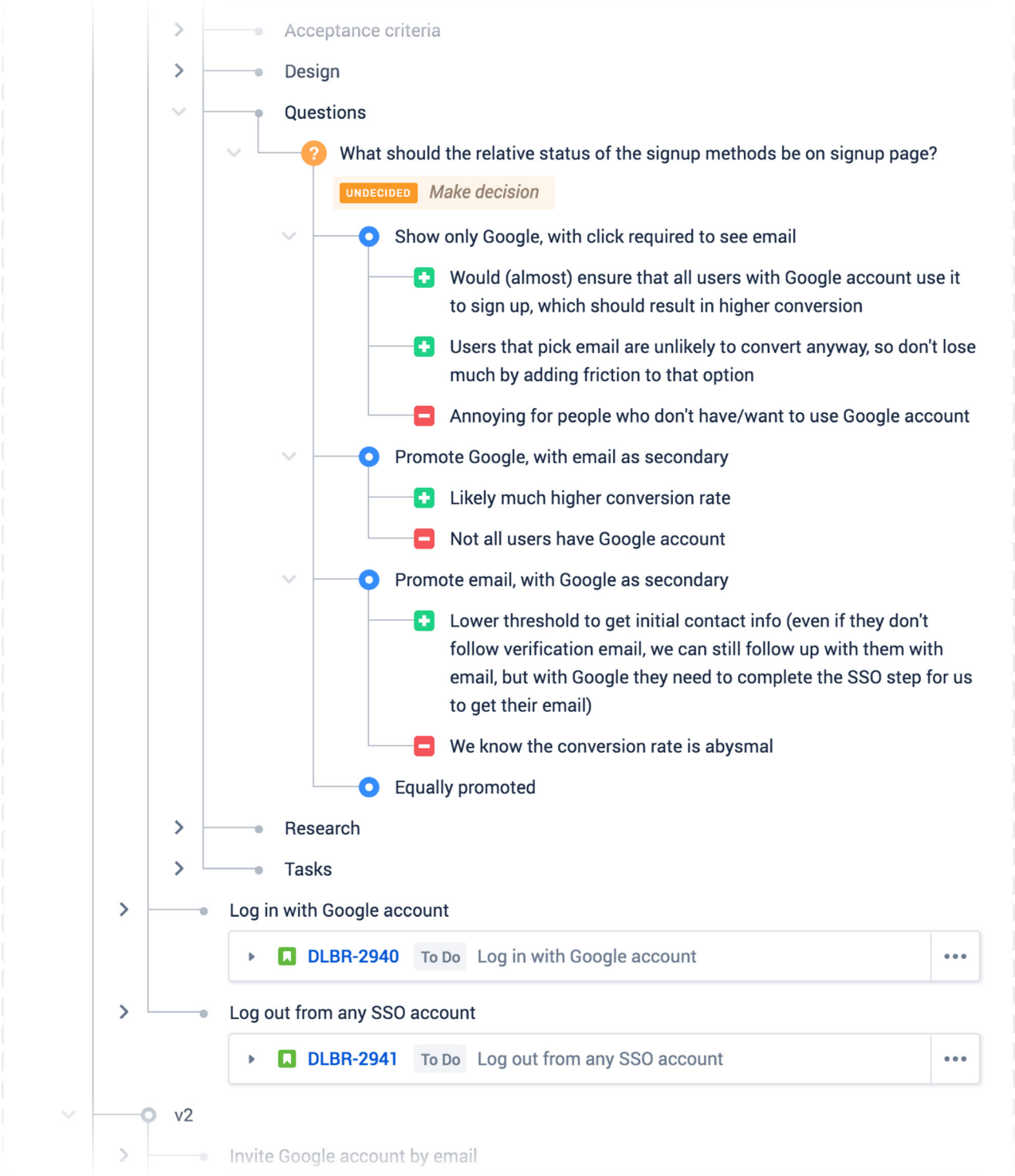
- Would require us to handle unverified user rights (e.g. maybe disallow them to invite others)
- Examples from other apps
- Measure of success
  - Signup completion rate, all the way from when users click "sign up" to when the are past the signup wizard and enter the app.
    - It is currently at 5%, which is not nearly good enough.
    - After this improvement, 15% should be considered a success.
    - The reason it may not be much higher, is that we have observed a ~50% obviously fake emails from the first step, and it is not feasible that these will be affected.
- Rollout plan
- Stories
  - User Story Map

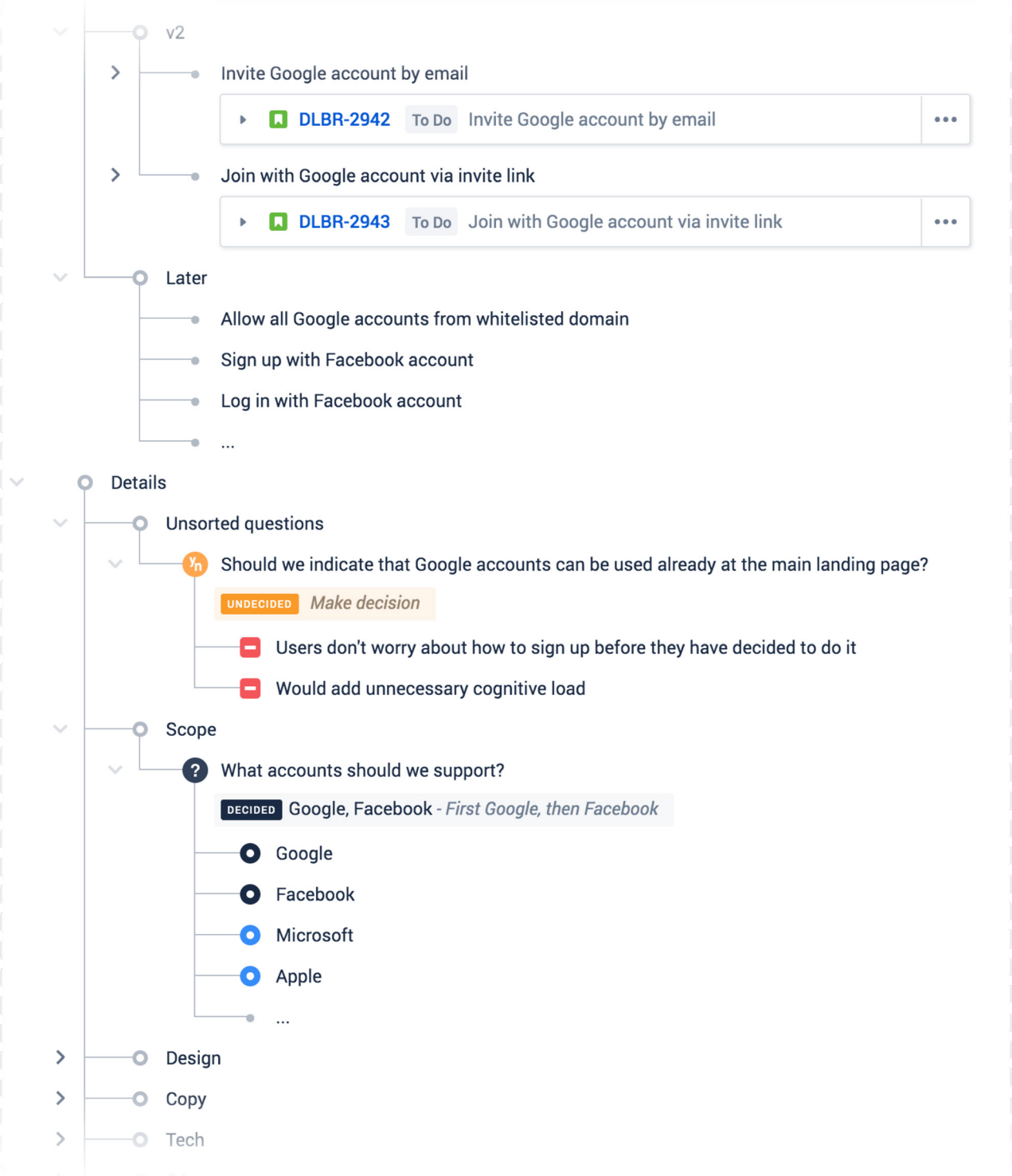


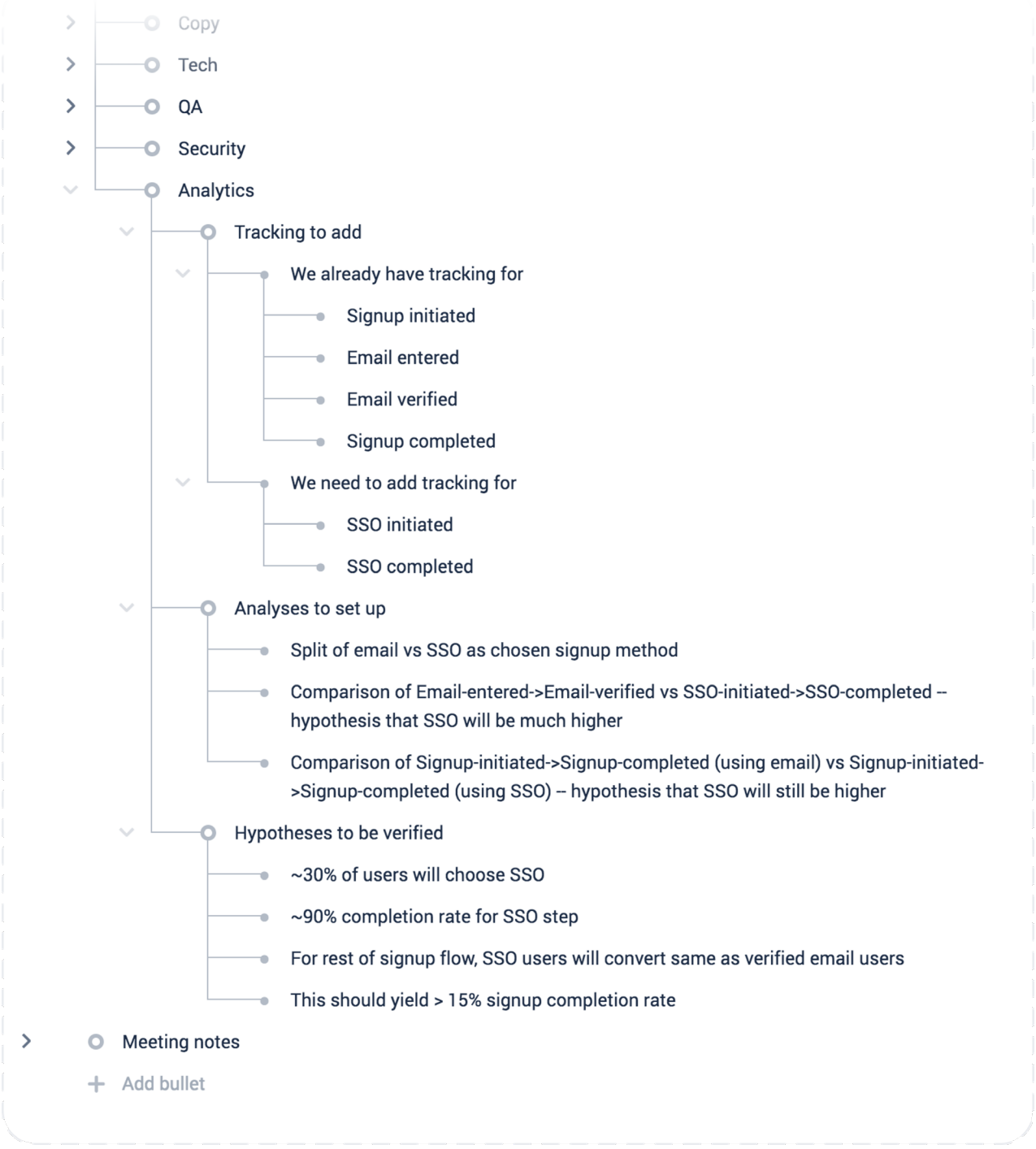
  - v1
    - Sign up with Google account



      - Acceptance criteria
      - Design


Acceptance criteria
Design
Questions
What should the relative status of the signup methods be on signup page?
UNDECIDED
Make decision
Show only Google, with click required to see email
Would (almost) ensure that all users with Google account use it to sign up, which should result in higher conversion
Users that pick email are unlikely to convert anyway, so don't lose much by adding friction to that option
Annoying for people who don't have/want to use Google account
Promote Google, with email as secondary
Likely much higher conversion rate
Not all users have Google account
Promote email, with Google as secondary
Lower threshold to get initial contact info (even if they don't follow verification email, we can still follow up with them with email, but with Google they need to complete the SSO step for us to get their email)
We know the conversion rate is abysmal
Equally promoted
Research
Tasks
Log in with Google account
DLBR-2940
To Do
Log in with Google account
Log out from any SSO account
DLBR-2941
To Do
Log out from any SSO account
v2
Invite Google account by email


v2
Invite Google account by email
DLBR-2942 To Do Invite Google account by email
Join with Google account via invite link
DLBR-2943 To Do Join with Google account via invite link
Later
Allow all Google accounts from whitelisted domain
Sign up with Facebook account
Log in with Facebook account
...
Details
Unsorted questions
Should we indicate that Google accounts can be used already at the main landing page?
UNDECIDED Make decision
Users don't worry about how to sign up before they have decided to do it
Would add unnecessary cognitive load
Scope
What accounts should we support?
DECIDED Google, Facebook - First Google, then Facebook
Google
Facebook
Microsoft
Apple
...
Design
Copy
Tech


Copy
Tech
QA
Security
Analytics
Tracking to add
We already have tracking for
Signup initiated
Email entered
Email verified
Signup completed
We need to add tracking for
SSO initiated
SSO completed
Analyses to set up
Split of email vs SSO as chosen signup method
Comparison of Email-entered->Email-verified vs SSO-initiated->SSO-completed -- hypothesis that SSO will be much higher
Comparison of Signup-initiated->Signup-completed (using email) vs Signup-initiated->Signup-completed (using SSO) -- hypothesis that SSO will still be higher
Hypotheses to be verified
~30% of users will choose SSO
~90% completion rate for SSO step
For rest of signup flow, SSO users will convert same as verified email users
This should yield > 15% signup completion rate
Meeting notes
Add bullet

The approaches seen in the example

Let's briefly go through how each of the four approaches from the book can be seen in the example document.

Drive towards impact, base it on research

The link to the overall focus of the business is clear in the problem statement. Research in the form of both quantitative data and qualitative feedback is included (albeit collapsed for brevity). The link between the business impact (signup conversion) and the customer outcome (frustration in signup flow) is also clear. Beyond that, it is also elaborated what success looks like, and how to evaluate whether success has been reached.

Embrace user stories as a shared language

The epic is expressed as a set of user stories. To put the stories into context and to help prioritization/slicing, the stories are projected on a user story map. For each of the stories, the conversation continues with more details in the document, culminating in a set of acceptance criteria (although only shown for the one user story that is expanded).

Create a shared understanding

The details are written in a document at the epic-level, with the template in the book as the starting point. Although rather difficult to illustrate, one can imagine that the document evolved from basically only a problem statement, with more details added over time, and with a few things that need to be decided and closed down before deploy. The fact that things can be collapsed to avoid cognitive overload illustrates that an outliner has been used.

Expose micro-decisions through explicit questions

Although leaving much of the collapsed content to imagination, there are a few visible non-obvious questions that have been posed explicitly and directly in the relevant part of the document (or in the questions to be sorted section). For some questions, decisions have been made. For some questions, rationale has been added. The questions, decisions, and rationale have been clearly marked.

I want your example epic documents

I hope you found the approaches and example both insightful and thought provoking!

The research that led to this book is still very much ongoing and we are always looking to learn more, and so any feedback is greatly appreciated.

The next step in the research will be to elaborate with more examples and anecdotes on blog.delibr.com. Therefore I would especially appreciate:

Examples of what your epic documents look like, ideally with comments on what affected the way you designed the document (e.g. the context of your company and the process that they fit into).

Anecdotes of how you work that relate to the approaches in the book (in the form of stories, quotes, screenshots, etc.), either in agreement with the book or rationale for why it is better to work another way.

Please send them to info@delibr.com.

Hopefully that can trigger some good discussions.

Acknowledgements

As this book is far from the work of me alone, I would like to extend my gratitude to the people who made the book possible.

Starting with the fact that you read this book at all. It is probably due to the work of **André Dantorp**. He helped connect the dots between the people involved and to reach out to the people who could help spread the book.

In case you wondered how come the book looks so great, it is thanks to **Annelies Clauwaert** who designed the book, **Raluca Mitarca** who worked together with Annelies to create the illustrations for the book and **Holly Ovenden** who designed the cover. I don't know if you noticed, but the illustrations for each chapter actually illustrate that chapter, using an elephant as a metaphor for an epic, going through the book. I was amazed by what they came up with!

I would like to thank **Jeff Patton** for the great foreword, and also for a lot of inspiration (if you read his book User Story Mapping you have probably noticed quite a few concepts carried over into this book). I would like to thank **Matt LeMay, Jonas Tellander, Reza Farhang, and Johan Östling** for writing so supportive comments about the book.

If you would compare my first draft of the book with what you have in front of you now, well... it would not be nearly as good. Over the course of several months I have received so much great feedback. Every time I thought I was getting close, I would get a new round of

feedback that seemed to amount to "if you could just turn the book around 15 degrees", and so I would go back and re-write big parts of it. I have to admit, it was at times really painful, but I am also really grateful for the feedback, as it made the book a lot better. For going deep into the book and providing invaluable feedback, I would like to thank: **Deepro Basu, Farid Bonawiede, Johan Bouzi, David Božjak, Staffan Eketorp, Thomas Eriksson, Reza Farhang, Sam Feldman, Jens-Fabian Goetzmann, Petter Hallman, Peter Hilton, Mikael Holmquist, Elisabeth Kamor, Sun Kim, Matt LeMay, Matt Lerner, Hiram Vazquez, Alexander Wilson, and Erik Wenneborg.**

To conclude my rant of gratitude, I would like to give a big thanks to the people who let us interview them. But before there was even a draft book, there were interviews, lots of them, and really good ones. All-in-all, André and I held more than 300 interviews. The basis for the book are the insights gained over time by conducting or watching these interviews. Big thanks to:

> **Saureen Adani**, Head of Product at Sarva
>
> **Brittany Arnett**, Sr. Product Manager at FormAssembly
>
> **Bella Arutyunyan**, Product Manager at Service Titan
>
> **Hamza Asif**, Product Manager at Kuali
>
> **Guy Assedou**, Director of Product at MobileCause
>
> **Paige Baeder**, Product Manager at PatientPing
>
> **Saket Bajoria**, VP and Product Management at Lucideus Inc

Mandar Bandekar, Sr. Product Manager at Ooma

Armen Bandikian, Lead Product Manager at Liberty Lending

Christi Banister, Sr. Business System Analyst at Red Hat

Deepro Basu, Sr. Product Manager at CloudBees

Walter Bellante, Product Owner at Vulog

Matt Bernstein, Product Manager at GrubHub

Sumit Bhat, Sr. Product Manager at Symantec

Jola Bolaji, Product Manager at Tic Toc Games

Gill Bordeaux, Founder/VP of Engineering at Smart Bumper Sticker

Zack Boyd, Sr. Product Manager at Engie Impact

Mark Brierley, Product Manager at Buildium

Karlene Cabison, Product Owner at Jigsaw Puzzle

Matt Cannavale, Product Manager at Muller Systems

Manny Cercado, Technical Product Manager at Doximity

Nick Chambers, Sr. Product Manager at Keet Health

Piyush Chandra, Director of Product, AI at Nauto

Angela Chou, Head of Product at Vevo

Andy Chu, Head of Product at Anagram

Eric Chu, Sr. Product Manager at Zocdoc

Michael Clinton, VP of Product at C-Trax

Prajna Cole, Project Manager at Concetric Sky

Michelle Cong, Product Owner at Teletracking

Tom Connard, Co-founder at 3D Source

Leandro Cordeiro, Product Marketing and GTM Lead at Intel

Jay Cosgrove, Sr. Product Manager at Digital Scientists

Brandon Crowfeather, Technical Product Manager at Clover Health

Kapil Datar, Product Manager at First Tek Inc

Matt Day, Sr. Product Manager at Lob

Richie Decena, Agile Product Manager at Beachbody

Fenil Dedhia , Technical Product Manager at Agolo

Aneri Desai, Sr. Project Manager at Unum

Marc Devens, Director of Product at BrandVerge

Meg Donchak, Product Producer at Facebook

Sarah Doyle, Product Manager at Lowe's Companies, Inc

Neil Dutta, QA Analyst at Namely

Robert Eisenstein, IT Director at HomeRun Homes

Mohamed Elsaman, Product Manager at Johnson & Johnson

Kyle Evans, Head of Product at Clearlink

Ashley Faggianelli, Sr. Product Manager at General Assembly

Sachin Fatnaney, Lead engineer at Ipreo LCC

Sam Feldman, Sr. Product Manager at TripAdvisor

Andy Ferris, Sr. Product Manager at Igneous Systems

Gary Fox, Product Manager at Bosch

Don Francis, Sr. Product Manager at Omadi

Anthony Frank, Product Manager at Watermark Insights

Anthony Fuger, Product Manager at Watermark Insights

Kate Gallagher , Sr. Product Manager at ThriveHive

Vivek Gandhi, Product Manager at Google

Catherine Ganim, Product Manager at Oyster Tracker

Sarah Georgini, Product Manager at Harmon HVAC

Victoria Golden, Director of Client Services Engineering at NewsCred

Mauricio Gómez Aguiñaga, Sr. Software Engineer at Uber

Joshua Guenther, Software Developer at VAE, Inc

Ajaya Gupta, Director of IT and author of "Make Outstanding Things Happen with Center of Excellence"

Nyasha-Harmony Gutsa, Head of Product at Occupier

Peter Haendler, Product Manager at Safe Fleet

Mark Hammel, Director, Product Managment at NAVEX Global Inc

Vishwas Hegde, Sr. Product Manager at TuneIn

Matt Helmer, Product Manager at Abra

Jordan Helzer, Software Development Manager at Infuse Medical

Rebecca Higgins, Director of Product Management at Nav

Kim Hilton, Sr. Product Manager at Abbott

Kate Hoffman, Director, UX at Air Arabia

Eleanor Hughes, Product Manager

Monica Huynh, Sr. Consultant at EY

Lizzie Jaeger, Lead Product Manager at Cultivate

Gourav Jain, Director Client Services at FIS Global Position

Pragya Jalan, Technical Lead at One Door

Zachary Jones, VP Engineering at Tripple Point Liquidity

Julian Jung, Product Manager at Remind.me

Andrew Kao, Head of Core Product at The Black Tux

Jesse Kaplan, QA Engineer at Consultant

Mary Ellen Kelleher, Director of Product at Smokeball

Gordon Kennedy, Product Manager at Wallmart

Michelle Kernan, Product Manager at TATA Consultancy Services

Sunaina Khanna, Sr. Product Manager at The Enrollment Management Association

Nidhi Khator, Director, Product Management at Incedo

Sun Kim, Sr. Product Manager at Kabbage Inc

Rob Kirkpatrick, Director of Marketing at BombBomb

Wing Kuang, Lead Technologist at Booz Allen Hamilton

Adam Kuhn, Product Manager at Aperio Group

Katarina Lackner, Growth Product Manager at Adobe

Yvette Lapompe, Sr. Product Manager at IAS

Douglas Lerch, Growth Product Manager at Cybrary

Matt Lerner, Director of Product at MindBody

Frederic Levesque, Product Manager at NetDocuments

Casey Lewis, Product Manager at Hopper

Simon Lewis, Product Manager at Residently

Minerva Li, Multi-Solution Product Manager at Adobe

Larry Liao, Project Manager at Velvetchrome Assets Review

Thomas Liccardi, Product Manager at Fullbay

Denise Lo, Enhanced Product Solutions at Knowable

Prasant Lokinendi, Sr. Product Manager at Turo

Stephanie Long, Sr. Product Manager at Adobe

Mark Lorence, Lead Product Manager at Onovative

Raymond Lou, Sr. Product Manager at The Wing

Michael Louie, Product Manager at Reddit

Charis Loveland, AI/ML Business Development Manager at Amazon Web Services

Donovan Lowkeen, Software Engineer at Endpoint Closing

Abe Maali, Technical Product Manager at Groupon

Courtney Machi, VP of Product at Andela

Chirag Madnani, Product Manager at Ad tech

Mitchell Maffeo, Product Manager at Kyruus

Christian Manuel, Sr. Software Engineer at The Learning Corp

Mark Mastrangelo, Product Manager at Adobe

Etienne Maugain, Sr. Product Manager at SurveyMonkey

Amanda Maynard, Sr. Business Systems Analyst at Harvard Pilgrim Health Care Institute

Jennifer McDonald, Director of Product Management at Terradatum

Gary McGuire, Software Engineer at KUBRA

Fabian Meier, Product Owner at Recommind

Zak Middelmann, Product Manager at Greenhouse Software

Max Milhan, Sr. Product Manager at Dispatch Technologies

Dmitry Miller, Lead iOS Engineer at Holonis Inc

Vikas Minocha, Solutions Architect at ADG Tech Consulting

Pashmeen Mistry, Product and Go-To-Market Strategist at Amazon Web Services

Sarah Morgan, Director of Product Management at EverTrue

Dwight Mouton, Product Owner at Advance Auto Parts

Talia Moyal, Director of Product, Marketing at Lightstep

Aravind Muchakhandi, Founder at SIMPLIN

Patrick Muggler, Product Manager at Continental

Jason Nelson, Sr. Marketing Director at Bluebird botanicals

Lan Nguyen, Product Manager at NerdWallet

Chrysanthos "CJ" Nicholas, Product Manager at Stensul

Heather Novak, Sr. Manager of Customer Solutions at Asurion

Joe O'Connell, VP Product at Moved

John O'Donnell, Sr. Product Owner at Trifecta Clinical

Michelle O'Keefe, Sr. Product Owner at Medidata

Jeff Orange, Sr. Product Manager at GoPro

Aaron Orr, Sr. Product Manager at Cricut

Paolo Padiernos, Group Product Manager at Policygenius

Mizue Parrott, Product Lead at PollyEx

Randy Patterson, Software Product Manager

Steve Petroff, Sr. Product Manager at Turvo, Inc

AlexAnne Pitts, Product Manager at BMW Group

Kurt Prenger, Software Engineering Director at Ascendum

Narsi Raghunath, Product Manager at Couchbase Cloud

Amit Rajput, Product Strategy Advisor at mipOS

Sergio Reyes, Product Owner at

Patrick Rice, Product Owner at Equifax

Carolyn Ripley, Product Manager at CyberGrants

Cara Rizzo, Sr. Product Manager at Leafly

Naren Roy, Sr. Product Lead at Shutterstock

Dennis Ryu, Sr. Product Manager at Olo

Donya Sabaghi, Product Manager at SalesLoft

Tyler Sanchez, Sr. Product Manager at Brightwheel

Sam Schmitz, Product Manager at Samcart

Michael Schneider, Sr. Software Engineer at Amazon

Liz Schoenecker, Product Manager at Beautycounter

Jeff Schrank, Programmer Analyst at CGM, LLC

Alexa Schulte, Sr. Product Manager at Hint Health

Misha Shah, Cheif Operating Officer at Think Latitude Inc

Brandon Shiaw, Sr. Product Manager

Vikram Shivashankar, Sr. Director, Market Solutions at Bidgely, Inc

Hilary Short, VP, Brand Development & Experience
at Commonwealth Financial Network

Jason Shugars, Head of Global Technology Partnerships at Blueshift

Bharat Singh, Director of Product Management at HG Insights

Kanwaldeep "KD" Singh, Sr. Director User Experiece at Clarivate-DRG

Gurmeet Singh, Sr. Product Manager at Optum Insigh

Jasti Smythe, Scrum Master at Elektrobit

Katie Spires, Sr. Product Manager at Wine Direct

Heather Staudt, IT Product Manager at American University

Maggie Stewart, Sr. Product Manager at OJO Labs

Christopher Sykes, Agile Project Manager

Ian Templin, Product Manager at StrongMind

Goeffry Thomas, Technical Product Manager at Currency

David Turell, Sr. Manager at Harvard Business Publishing

Madison Unell, Sr. Product Manager at Ribbon

Gabe Vasquez, Sr. Director of Marketing at World Education Services

Hiram Vazquez, VP of Product at ReloQuest

Ash Vemulapalli, Sr. Product Manager at PubMatic

Melissa von Holstein, Sr. Product Manager at The Zebra

Benjamin Walker, Business Process Strategy Manager, Vice President at BBVA in the USA

Carol Walport, Product Manager at Google

Katie Walters, Product Manager at Digital Scientists

Jack Wheeler, Product Manager at The New York Times

Kas Williams, Technical Product Manager at Otis Elevator Company

Benjamin Wilson, Application engineer at FDB

Amelie Wisniak, VP of Product at Pypestream

Donna Yee, Product Manager at The Know Worldwide

Shany Yeshanov, Product Manager

Susan Yu, Customer Success Manager at Reveleer

Laurence Yudkovitch, Product Manager at TeraMedica

Neda Zaman, Marketing Manager at Imerys

Sam Zimmerman, CTO at COVID Safe Paths

...and many more.

ABOUT THE AUTHOR

Book backstory

Hi, I'm Nils Janse. I'm an engineer by education[27], a management consultant by training[28], and an entrepreneur by occupation[29].

But to be perfectly honest, my motivation for writing this book isn't only to help product managers, but also to save democracy.

> *I believe that product managers are leaders among knowledge workers. By reading this book they can become better at making deliberate decisions in close collaboration with other people. Because this will make them expect the same of others, it will make them, and the other knowledge workers they inspire, better citizens. Over time, I hope this will help the survival of democracy.*

27 MSc Industrial Engineering / Mechatronics, the Royal Institute of Technology (KTH)
28 6 years at McKinsey, incl. several projects involving teams working with product development
29 Several startups, incl. traffic control system with app used by all subway drivers in Stockholm.

Seems like a big leap? I'll break it down into seven steps:

1. Passion - I became super passionate about democracy by traveling, experiencing and comparing the top and the bottom out of The Economist Democracy Index[30].

2. Problem - Yet democracy seems to be in trouble: political debates on TV often resemble kindergarten disagreements, not to mention debate on e.g. Twitter.

3. Vision - What if they sounded like: "Ok, so on the question of A, although I disagree with your proposed alternative B, I do think you have a strong argument for it with C. However, I believe in alternative D, as I think the argument E is even stronger, because of fact F."

4. App attempt 1 - In 2012, I tried to develop an app directly for politicians to make the vision come true. I failed miserably, as they were motivated by winning over the opposition, not in having balanced debates.

5. App attempt 2 - Then we tried to make an app for collaborative decision-making for business. We had some early success, but failed the toothbrush test.

6. App attempt 3 - Our best use cases were PMs facing decisions on questions with a few alternatives, evaluated by several people - perfect for the app. To pass the toothbrush test we

30 Visited from top: Nordics & the Netherlands, from bottom: Chad & North Korea

made a major pivot into an app for writing feature documents, where the app to facilitate the decision-making became one feature. This worked much better.

7. Research - To really understand the use case and how PMs can become more effective in their job, I spent the past couple of years interviewing hundreds of product managers, and working more closely with many of them. This book is the result.

To sum up, with this book I attempt two things. First, to codify what I have learnt and make it useful to product managers. Second, to promote skills and behaviours that it would be great if citizens expected of their democratically elected leaders. In a democracy, over time, people tend to get the leaders they deserve. This may take decades, but when I thought about what to do, this is the best I could come up with...