Intercom on Product Management

Intercom

Intercom on Product Management

Some helpful thoughts on the challenges of building
software products from the team at Intercom.

Our software enables internet businesses to see who is using
their product and makes it easy to communicate with them
through email and in-app messages
www.intercom.io

We also regularly share our thoughts on product management,
design, startups and the business of software.
blog.intercom.io

Cover and chapter illustrations: Quentin Vijoux
Hooked images in Chapter 1 are courtesy of: Nir Eyal,
Author of “Hooked: How to Build Habit-Forming Products” Blog at: NirAndFar.com

Readers on subway photo in Chapter 3 by Alfred Lui on Flickr

ISBN 978-0-9861392-0-8

TABLE OF CONTENTS

INTRODUCTION

At Intercom we believe a great product should be the first focus of every startup.

Product management - a job which has elements of engineering, marketing, research and project management - is key to building great products. It requires a relentless focus on cohesion; ensuring that how a product is designed, engineered, named, branded, and marketed are all united under a single vision. It is still a young and rather ill-defined discipline, due to the constantly changing nature of software products. Some argue it's not even a distinct discipline these days. In the startup world a lot of people become PMs by default rather than design.

Regardless of what route you came by, or what your current role is, if you are a product person we compiled these articles for you.

We've been writing about our product lessons and experiences on the Inside Intercom blog for just over three years. This short book connects the most relevant posts on product design and management from that body of writing so that others can hopefully fast track their development. Regular readers will note that we added a few sections to connect ideas or bridge gaps between articles.

We'll start by evaluating your current product, looking at what parts of it are being used and what areas are ripe for improvement. In Chapter 2 we'll look at the hard choices that surround building new features and why "no" is the most important word in the product manager's vocabulary. Chapter 3 will give you a checklist that every new feature has to pass and discusses how to roll them out. We'll finish by looking at how you get those new features used by your customers. Because at the end of the day shipping code means nothing unless it is used and valued.

Let's jump in.

CHAPTER 1

Evaluate your product

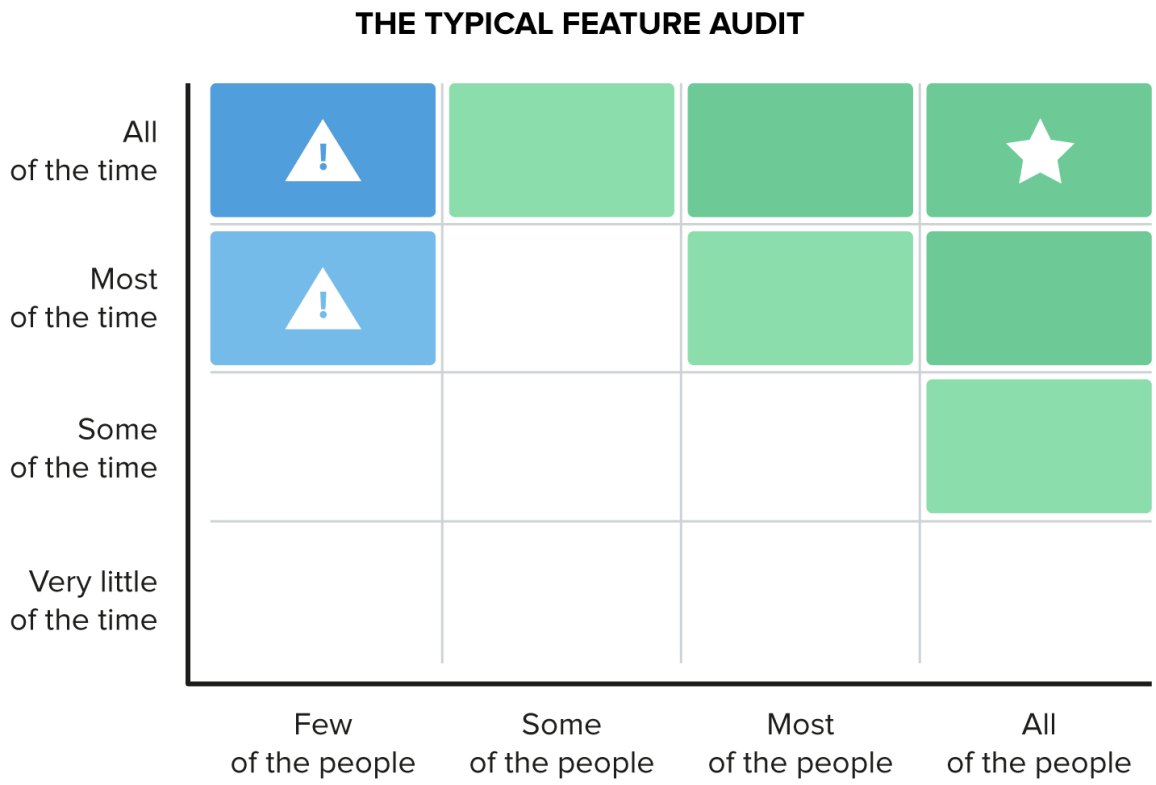
Let's start at the beginning. You're a product manager? A startup founder? CEO? CTO? Whatever your job title let's assume you're making the decisions about your product. What features to develop, when to develop them, how to get users to start using them - that kind of thing.

As product people we all have great hopes and dreams for our products and how they are going to fare once they hit the big bad world. You're probably filling notebooks full of ideas that will make you the next Uber, Slack or Facebook. But before you start creating a product roadmap that's going to lead to greatness it's time to take stock and see what's currently going on in your product.

What are people actually doing in your product?

Are all users using all the features in your product? Of course they're not. Let's start by talking about that.

When planning your roadmap, and where your team spend their time, it's useful to ask "how many people are actually using each of our product's features?". This is product management 101 and will probably take you a few minutes of SQL, or a couple of seconds of Intercom. A simple way to visualize feature usage is to plot out all your features on two axes: how many people use a feature, and how often. You'll most likely see trends like this one.

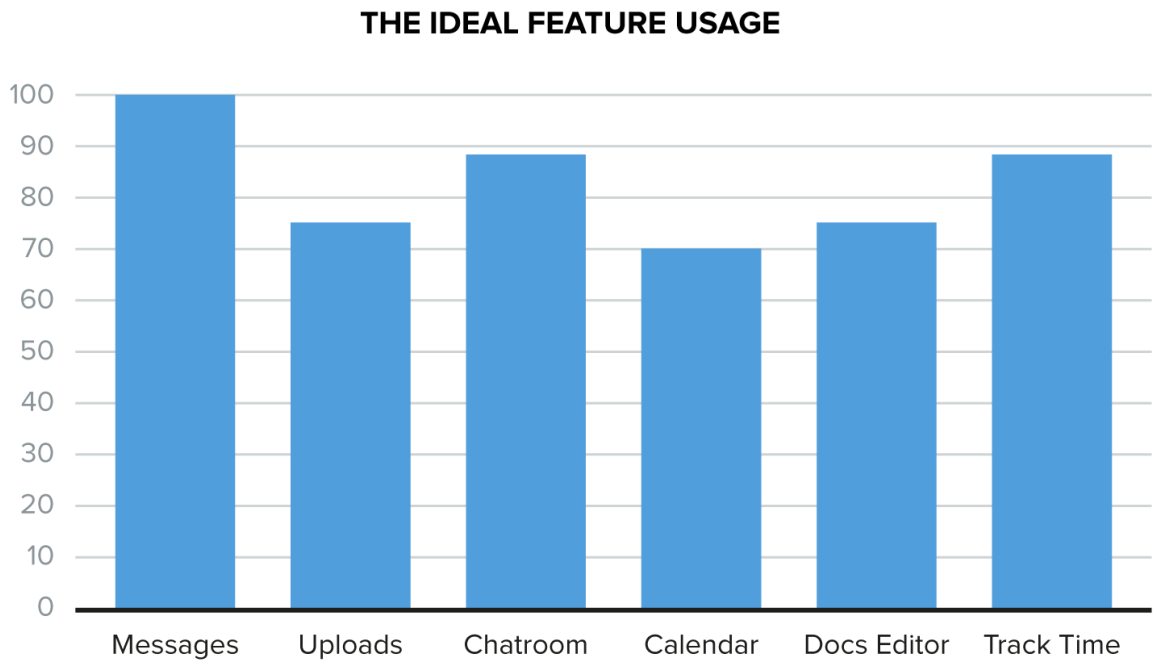


The core value of your product is in the top right area, up where the star is, because that's what people are actually using your product for.

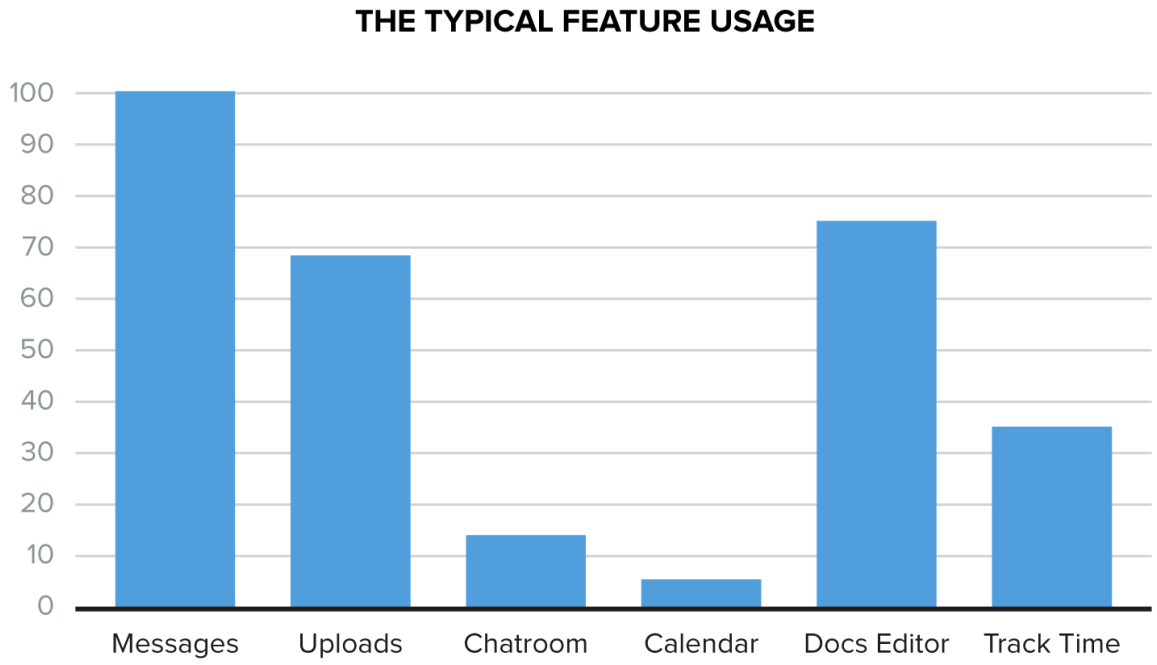
Sidenote: Exclude administrative features like account creation, password reset, etc. from this exercise. They're not relevant here. Also, exclude features that only certain users (e.g. enterprise customers) can access. They should be evaluated separately.

If you have features in the top left it's a sign of features with poor adoption. In other words, there are a small amount of customers depending on this feature, but most rarely touch it.

An even simpler way to think about it is this: What percentage of your customers or users have adopted each feature? You can do this with simple bar charts. Shown here is a dream product. The one we all think we're creating when we have Photoshop open. All my users are going to use and love all these features, right?



But as every product manager discovers, the messy reality of a product looks a lot more like this.



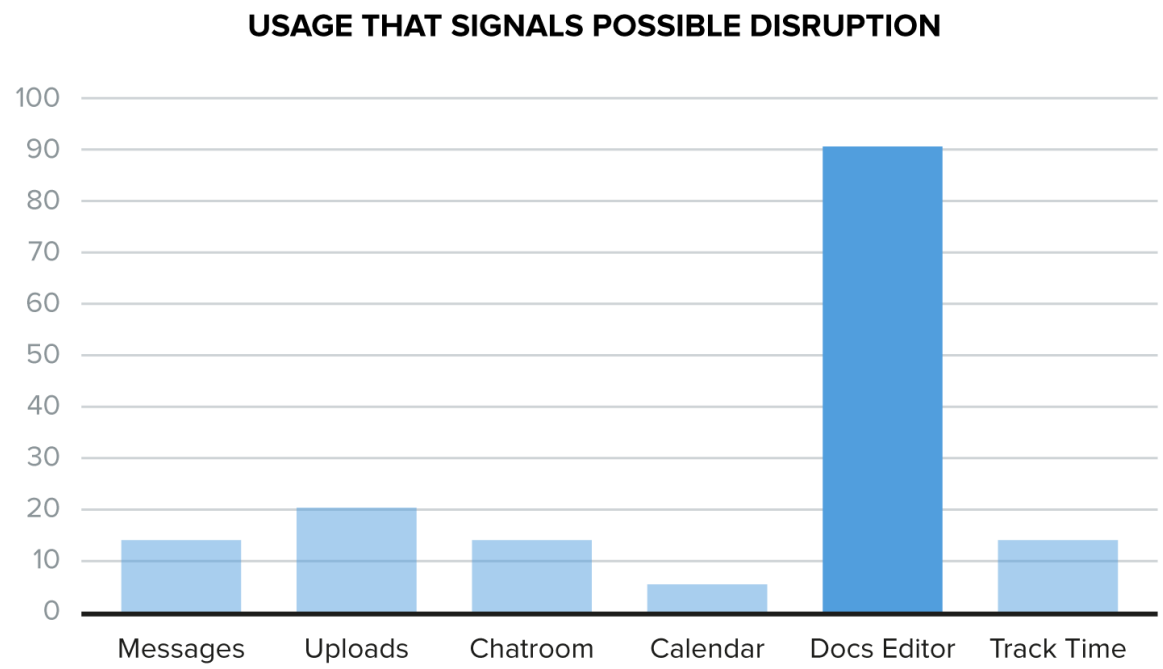
How did you get here? You built a product that had solid messaging, files and document editing features. These key features were highlighted on your marketing site, documented clearly with great screenshots, included in your product tour, and every new sign-up loved them.

But success can be a lousy teacher. You believed you could do no wrong and that you could build lots more.

So you added a chat room, but it didn't go so well, you kinda botched the launch of it. Then you added a calendar and that went worse. No one created more than one event and it's still not even mentioned on your marketing site. Now you've built this time tracking features that's kind of popular, but only with a certain type of user.

This is your product, you need to fix this.

A sidenote on disruption



If you're looking at this sort of feature breakdown, you have an excellent product for one precise workflow, but you've added all these other pieces no one

uses. Think of Hangouts inside Google+ as an example here, or primitive document editing (i.e. **B**U*I*) inside Microsoft Word.

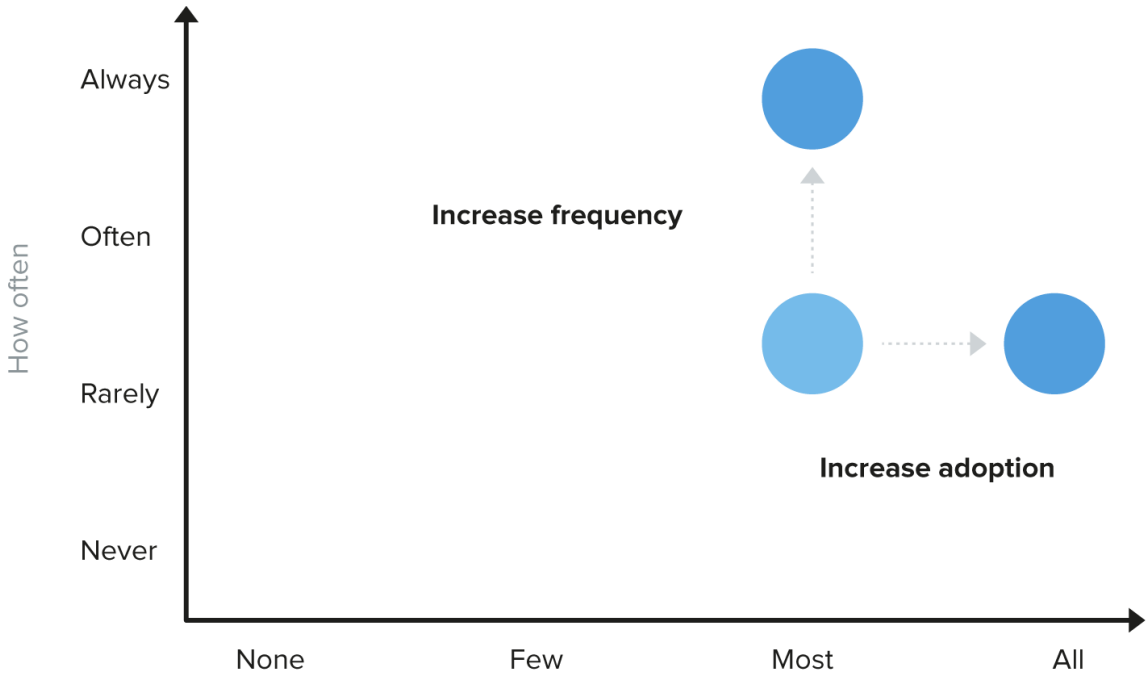
If you are looking at a chart like this you are vulnerable to disruption, in the true Clay Christensen sense of the phrase. Someone can build a simple product, focussing on that one key feature that's superior in just one way (cheaper, faster, collaborative, easier to use, mobile etc.), and you'll struggle to compete, because you're carrying all those other junk features around too.

What do you do with your feature audit?

For any given feature with limited adoption, you have four choices:

- Kill it: admit defeat, and start to remove it from your product.
- Increase the adoption rate: get more people to use it.
- Increase the frequency: get people to use it more often.
- Deliberately improve it: make it quantifiably better for those who use it.

Roughly, you can visualize it like this:



How to improve your features

Kaizen is the philosophy of continuous improvement. Web businesses searching to find product/market fit all follow some variation of *Kaizen* whether they know it or not.

Shipping code doesn't mean that you're improving anything. Similarly, you can make undeniable improvements to parts of your product and get no response or appreciation for it. It all comes to down to the type of improvements you're making.

The two most popular ways to improve a product are to add new features, or to improve existing ones. First we are going to look at ways to improve existing features before moving on to new features in the next chapter.

Improving existing features

You can improve an existing feature in three different ways:

- You can make it better **(deliberate improvement)**.
- You can change it so customers use it more often **(frequency improvement)**.
- Or you can change it so more people can use it **(adoption improvement)**.

1. DELIBERATE IMPROVEMENTS

This is when you know why customers use an existing feature and what they appreciate about it. A deliberate improvement seeks only to make it better in ways that will be appreciated by the current users. For example making it faster or easier to user, or improving the design.

Use deliberate improvements when: there is a feature that all your customers use and like, and you see opportunity to add significant value to it.

It's worth noting that deliberately improving a well-adopted frequently used feature is high risk high reward. As an example think about improving the editor in a blogging platform. Get it right, and every single users gets the bene-

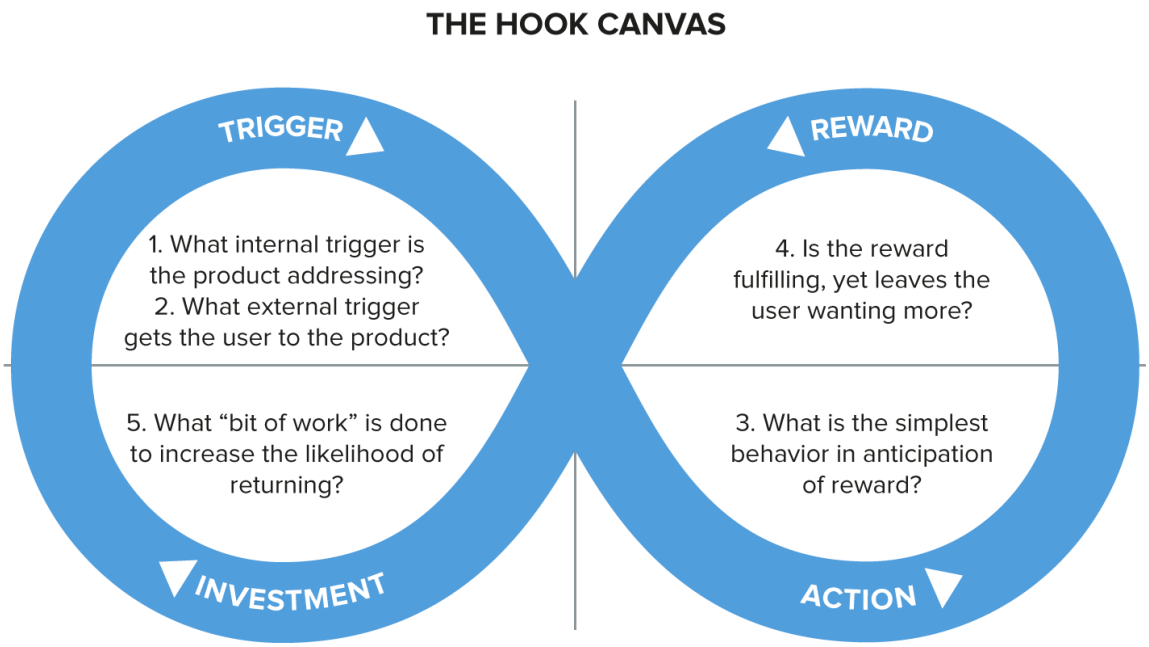
fit every time they use it. Get it wrong and you've broken the workflow of your entire userbase. High risk, high reward.

2. FREQUENCY IMPROVEMENTS

These are improvements that hope to get a customer to use the feature more often. Adding more items to an activity feed, or more options to a search tool means that people read it more often, or use it for more tasks each day. This type of improvement can turn a once-a-week feature into an every day feature.

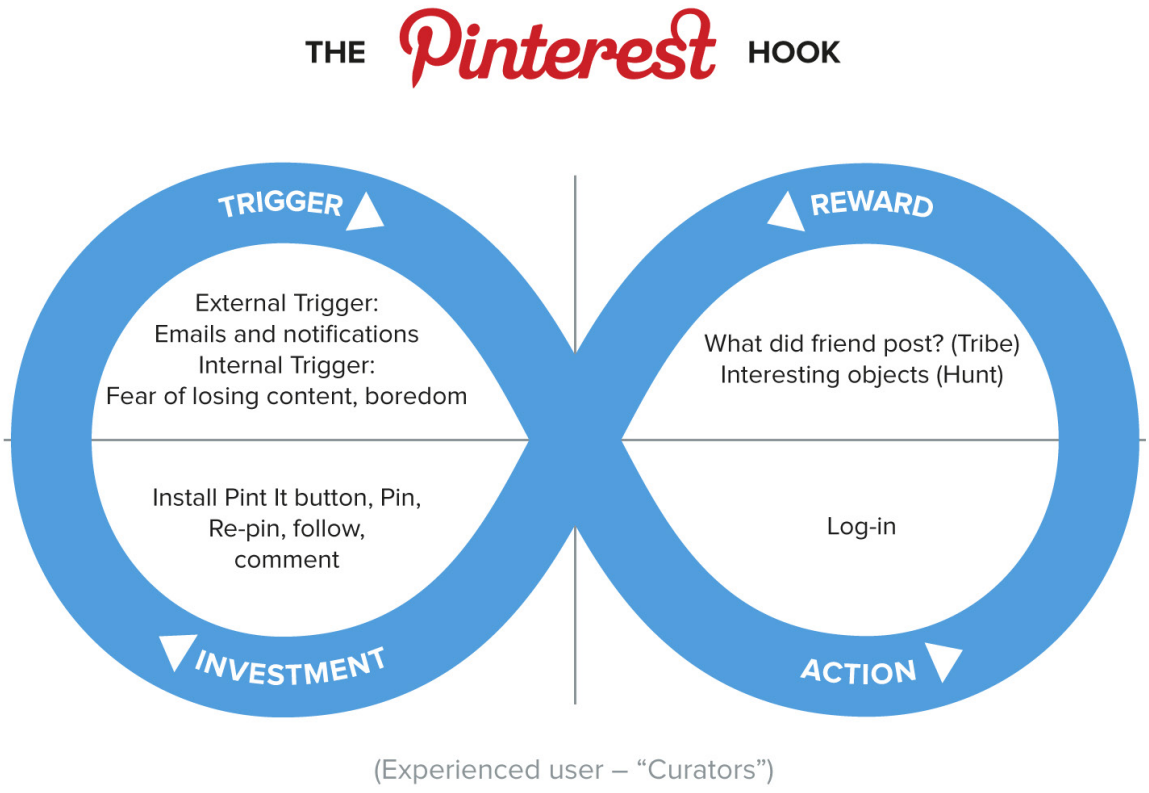
LinkedIn endorsements do this quite well. They added in these one-click multi-directional endorsements where you can easily (dare I say, accidentally) endorse four of your friends for skills they don't have, and accidentally connect to four more people as a result, which in turn creates more endorsements, more logins, and more single click broadcasts. A vicious circle.

What they're following is a pattern explained by Nir Eyal, author of Hooked, who says habits are formed from a repeated pattern with four key elements...



Trigger: the reason the user goes to the product (e.g. you received an email to say a contact had endorsed you for a skill).
Action: they take in anticipation of the reward (e.g. scroll, search, browse, etc).
Reward: the user gets from taking their action (e.g. seeing a beautiful Pinterest board).
Investment: the user makes which will plant the seed for more triggers (e.g. subscribe, pin, like, connect, etc).

As an example, Nir points to Pinterest's hook as the following:

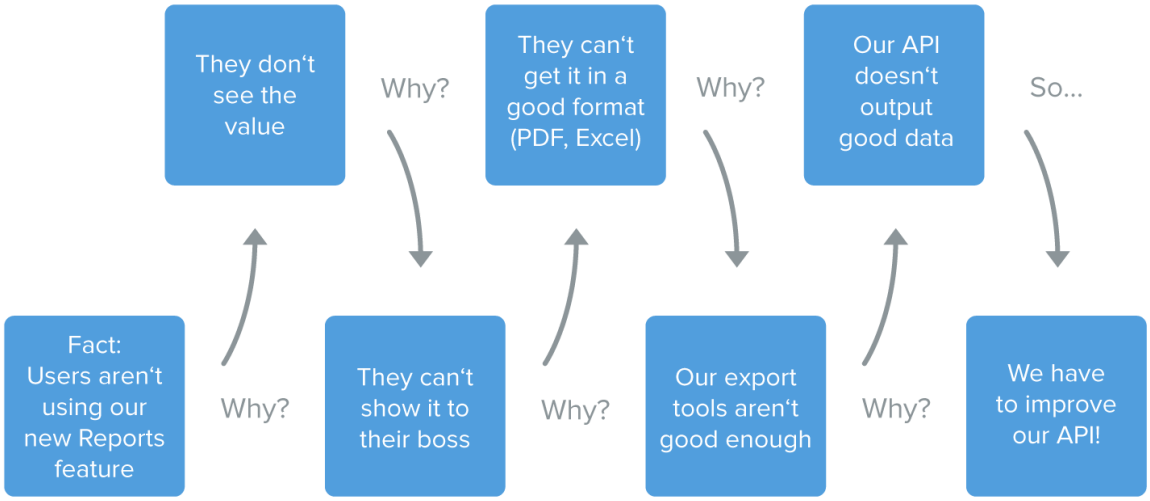


(Experienced user – "Curators")

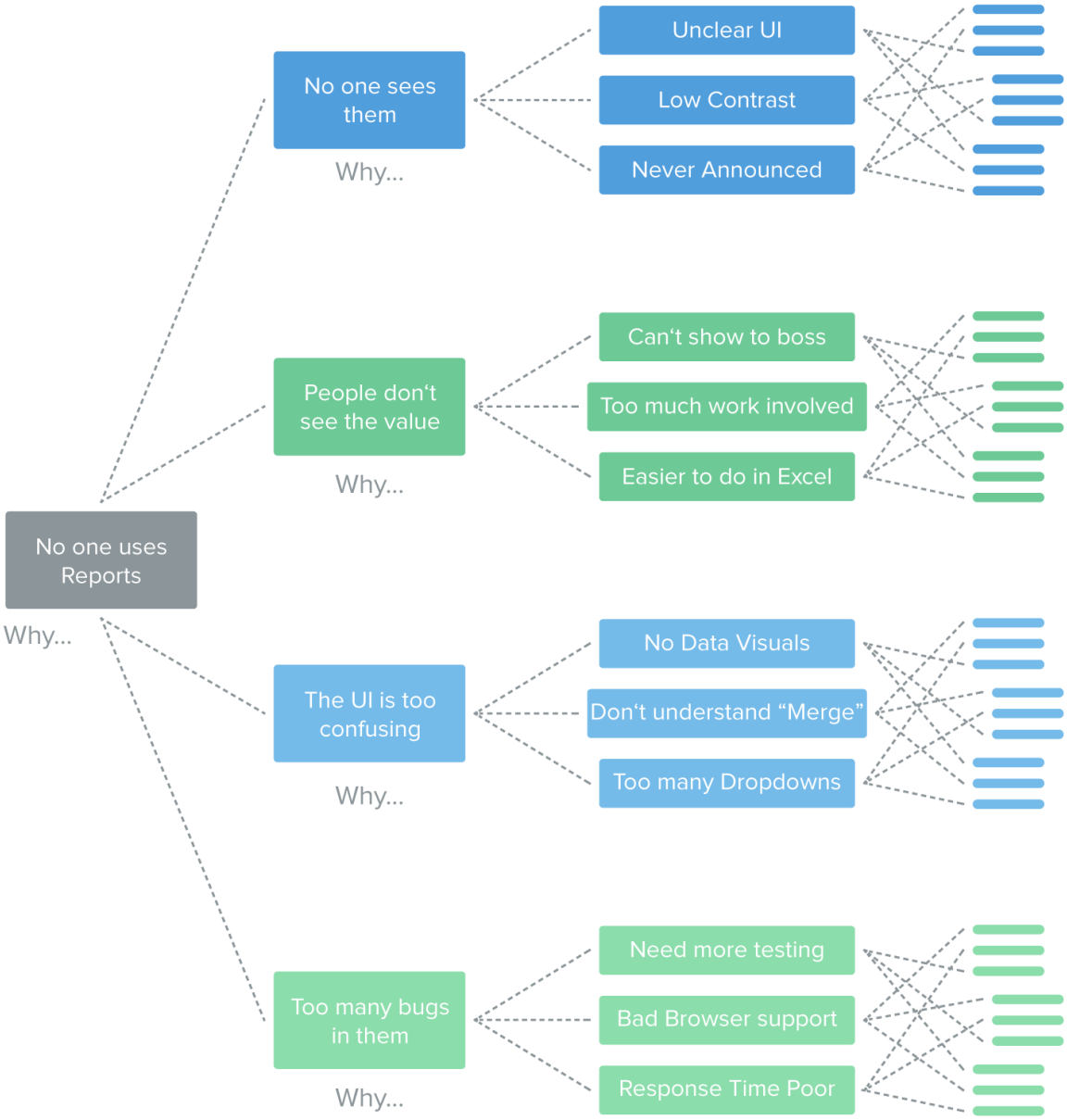
Use frequency improvements when: there is a feature that the majority of your customers use infrequently, and you believe that using it more would be of benefit to them. It's worth considering how your business will profit from this increased frequency. For example if Basecamp improve the frequency at which users create projects, they'll naturally profit, as that's how their pricing works. But there are lots of easy ways to gamify/hack feature usage that have no net benefit, or in some cases actually damage the core product offering. For example, LinkedIn's metrics for endorsements might look much better as a result,

but it's worth asking have they paid the price in credibility?

3. ADOPTION IMPROVEMENTS



Adoption improvements target those who don't use a feature. To get more people using it, rank and resolve the issues that are stopping them from using it. This is where the five whys technique is genuinely useful. You might have a situation around users not using your reports feature. Why? They don't see the value. Why? They can't show it to their boss. Why? They can't get it into a suitable good format. Why? Because our export tools aren't good enough? Why? Because our API doesn't produce good data. If you ask why enough times, eventually you'll work it out and get to the root cause.



Don't just talk to one customer because invariably things are more complicated than that. You'll find these blocking patterns over and over again. You can then resolve the key issues, the fundamental things that are actually blocking people from using all of your product.

Use adoption improvements when: there is an important feature that a good chunk of your users have yet to adopt, and you see some obvious integrations or changes that will make it easier for them to get on board.

When planning adoption improvements, always consider improvements

outside of the software too. Sometimes it's not about how the feature is designed or built, it's about how it's explained. Often users just need to know *why* or *how* to use a reporting feature. In those cases better product marketing and customer communication is how you solve it, not product tweaks.

Continuous improvements

In their early stages startups have advantages over the incumbents. They move quicker and adapt faster, without much technical debt, legacy features, compatibility issues, or high value customers restricting their movement. Sometimes this speed and agility can cause startups to pivot like headless chickens, rather than focusing on improving their product in meaningful ways. The product manager's challenge is two-fold; firstly, finding improvements that will benefit the business and its customers, and secondly, ensuring that these improvements don't get lost on a whiteboard somewhere, and actually make it out the door. Because if there's one thing that's true for startup web products, it's this: if you're not shipping, you're dead.

CHAPTER 2

When to say no to new features

There is a finite amount of improvement you can put into your existing product and feature set. At some point you are going to have to consider what new features make sense for your product and how you are going to introduce them. Facebook famously faced down a revolt from some of its earliest and most loyal college users when it introduced the News Feed. But would it ever have seen the kind of global growth it enjoyed if it hadn't faced down those users and shown them the value of the new features?

In this chapter you'll learn about creating a product roadmap, with a formula for predicting where your efforts are best placed. But perhaps most importantly for product managers you'll learn why "No" is the most important word in your vocabulary and why you should hear yourself saying it a lot.

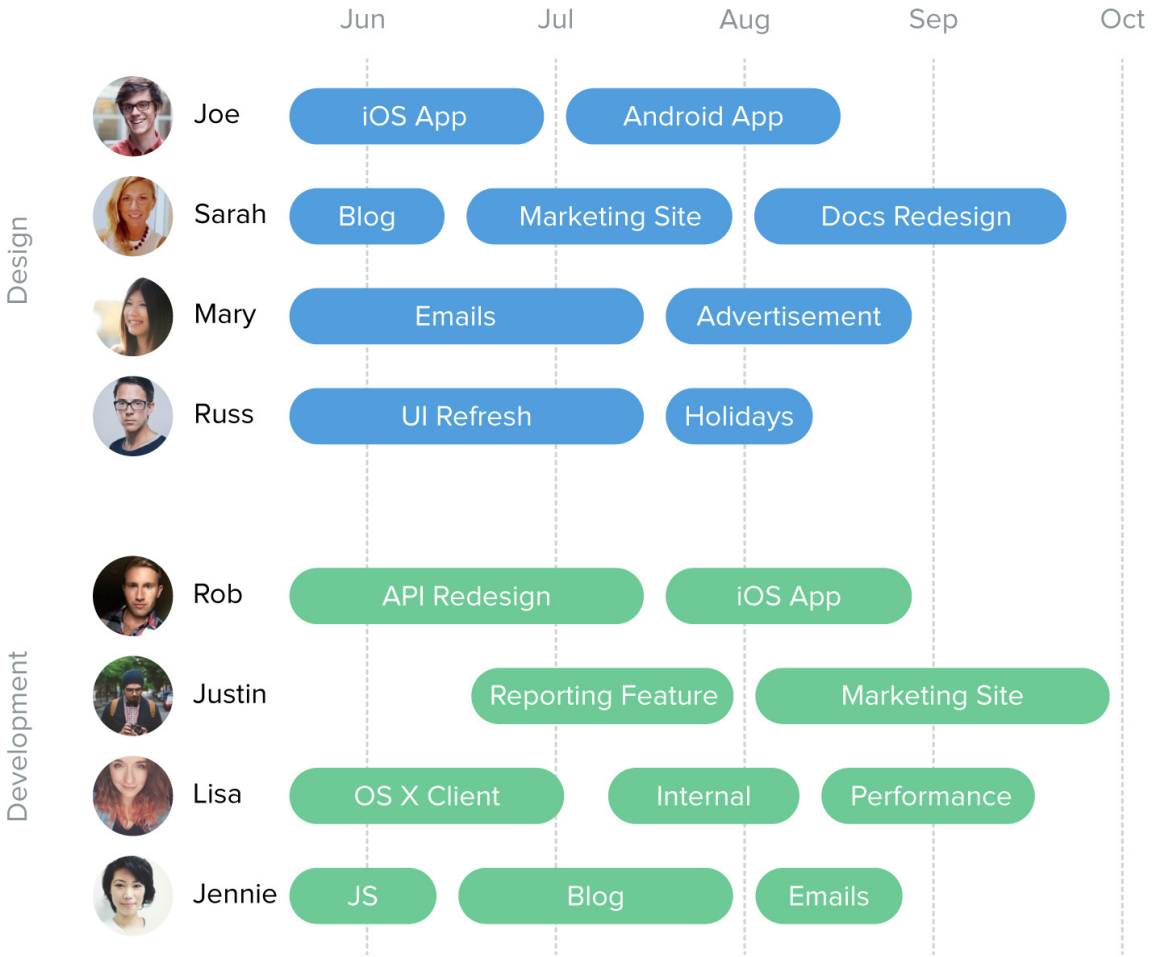
Adding New Features

New features expand the scope of the product, often making a big marketing splash, getting a version bump, and result in some press coverage. Often the fanfare attracts new customers and new use-cases for the product. Typically, new features are the only improvements that outsiders (i.e. non-customers) will ever hear about.

New features are risky. You have to be very confident they will be valued, as they're like children; you have support them no matter what.

Ask your customers "Would you like a (Calendar/TimeTracker/Gantt Chart)?" and they'll reply "Yes". It's a one-way "something for nothing" offer; why wouldn't they? They haven't had to make a trade-off between competing priorities. This leads to customers saying they want stuff that they don't really want.

Asking your customers "Would you rather that we made the product much faster, or that we added more labelling features?" and you'll get a different answer. Everyone values speed. So when planning new features it's important to understand the trade-offs at play.



Where do you suck? Where does it matter?

If you are going to build new features then you need to get them on your product roadmap - maybe yours looks something like the example above. A roadmap is built out of hard decisions. The bugs you must fix will fight with the features you must finish, the features your customers want will compete with the ones you know they need.

If you focus only on new features you'll build a product that is miles wide and inches deep. And if you focus only on repairs you'll never innovate, thus becoming irrelevant. Hard decisions indeed.

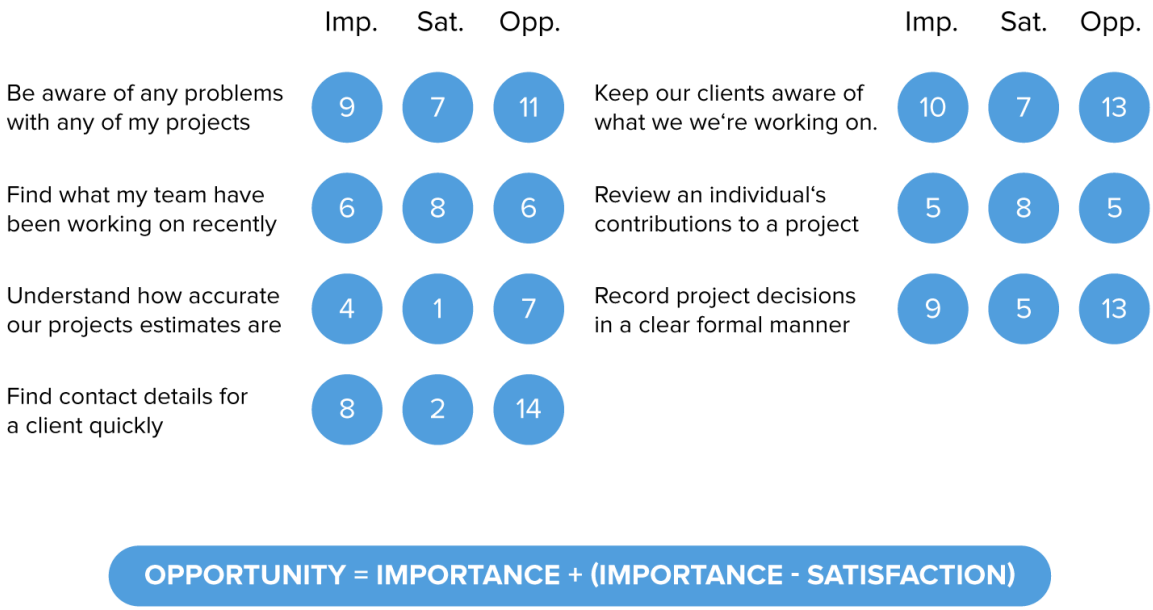
When you are focusing on improving your product, a great question to ask is

“Where do we suck, and where does it matter?”

To improve a product you focus on the parts that are both important and disappointing to customers. Both are required otherwise you’ll end up working on areas that no one cares about, or over-serving areas where you’re already more than good enough.

In his popular article, Turning Customer Input into Innovation, Anthony Ullwick proposes an opportunity algorithm which offers a practical way to plan a roadmap, taking importance and satisfaction into account.

Where the opportunities lie?



Ullwick’s simple opportunity algorithm cleanly identifies the shortcomings in a product, by getting customers to rank the jobs they need to do by how important they are, and how satisfied they are currently. The opportunity is then simply expressed as:

importance + (importance - satisfaction)

Sidenote: The bracketed element bottoms out at zero, over-serving a job doesn't reduce it's opportunity.

Aside from the simplicity of the formula, another nice trait is how it often highlights opportunities that would otherwise have gone unnoticed. Regularly the biggest opportunities lie in areas the product manager regards as being "complete", "bug free", "good enough" etc. Put simply: a minor improvement on an important task is almost always a larger opportunity than a big improvement on an ancillary one.

Here is where the 80/20 school of thought can lead product managers astray. The idea that 20% of the features will get you 80% of the value may well be correct, but it also means that on important tasks, you're giving customers a B-grade experience where it matters most. This point was made by Mark Zuckerberg talking about the early days of Facebook...

> You can't just 80/20 everything. There have to be certain things that you just are the best at. Things where you go **way further** than anyone else does, to establish this quality bar and have your product be the best thing that's out there.
>
> 
>
> **MARK ZUCKERBERG**
> CEO of Facebook

There's no right way to prioritize a roadmap, but there are plenty of wrong ones. If there are opportunities in existing product areas, and your roadmap ignores them in favour of new features, then you'll soon be that jack-of-all-jobs product.

Address your product's shortcomings, or someone else will.

Why you don’t add new features

Product managers, or founders fulfilling that role, have to be great at saying no. Not “maybe” or “later”. The only word is “No”.

Building a great product isn’t about creating tons of tactically useful features which are tangentially related. It’s about delivering a cohesive product within well defined parameters.

When your product gets traction, you’ll find yourself inundated with good ideas for features. These will come from your customers, your colleagues, and yourself. Because they’re good ideas, there will always be lots of reasons to say yes to them. Here’s 12 arguments that are commonly used to sneak features into a product:

1. But the data looks good

“We’ve tried this feature with a small group and engagement is off the charts.” Often this approach suffers from selective data analysis. Products are complex systems. What appears to be an increase in engagement is really just pushing numbers around from place to place. You might consider your labels feature a success based on its usage, but did you notice that that your tags feature is suddenly no longer popular? Even when the data is solid, and the increase in engagement is good, you still have to question whether it fits within the purview of the product. Add Tetris to your product and you’ll probably see a boost in engagement, but does that mean your product is better?

2. But it’ll only take a few minutes

The main problem with this argument is that the scope of work should never be a reason to include a feature in a product. Maybe it’s a reason to bump it up the roadmap as a quick win, but that’s a roadmap decision, not a product one.

Lots of bad ideas can be built quickly. Don’t be seduced. There are no small changes. Even the tiniest additions add hidden complexity that isn’t accounted

for in the “but it’s just 5 minutes” estimate.

3. But this customer is about to quit

This is feature blackmail. No customer can be more important than a good product. The road to consulting-ware is signposted “just this once for just this customer”. It leads to the perfect product, for just one customer, and even then their loyalty now depends on you doing what they say. Delivering extra value to one customer comes at the cost of taking value away from many others.

4. But we can just make it optional

This leads to death by preferences. Making features optional hides the complexity from the default screens in the interface, but it still surfaces everywhere else. The visible cost of this is a messy interface with lots of conditional design and heaps of configuration. The hidden cost is that every optional feature weakens your product definition. You become “a time tracker that can also send invoices and, sorta, do payment reconciliation. But not reporting. Yet…I think…I don’t know.”

5. But my cousin’s neighbour said…

This is the “appeal to the anecdote”. It is rife in consumer products, and in SaaS companies that can’t decide what precise jobs they do. Extrapolating from a tiny sample is an easy way to bypass years of experience, research, data, and behavior to make a statement that sounds reasonable. Saying “*my brother’s company use Google Analytics, they all use advanced segments*” is an easy way to make a case for advanced segments, bypassing key questions like:

- what your product actually does.
- whether your brother’s company are a good target customer.
- whether they actually use it or just say they do.
- and whether advanced segments are actually the right solution for what your customers are trying to do.

6. But we've nothing else planned

The devil makes work for idle product teams, and boy does he come up with some shitty features. The problem here is someone sees one or more engineers sitting idle and immediately rushes through a new feature to "keep 'em busy". Decisions are rushed and designs are cobbled together all in the name of avoiding idle time. This is a bad way to "improve" a product.

Instead of adding to technical debt here, there's an opportunity to pay some off. As anyone who has worked in a professional kitchen knows: "if you've time to lean, you've time to clean". Idle time is best used fixing bugs, cleaning up test suites, refactoring, etc. rather than derailing a product vision just to "keep the team productive".

7. But we're supposed to be allowed to work on whatever we want

This argument appeals to cultural pride. There are many big name companies that promise engineers they can build whatever they want and ship it. Usually this promise has one of two outcomes:

1. It's a lie told to attract engineers. This gets noticed quickly and falls apart. You can't fake culture.
2. It's true. The end result is a one-size-fits-none product full of half-baked ideas.

There's a difference between encouraging engineers to build things internally (a good thing) and letting people add features to a product bypassing product management (a bad thing).

8. But 713,000 people want it

Always beware when someone falls back to raw numbers to justify something. Any product with any amount of traction can make an emotive claim using numbers. e.g. "You could fill Dolores Park with people who have asked for Excel integration." Such a claim forces you to take off your product design hat, and be

one of the “people”. Are you really going to say no to all those faces?

You have to. Because the majority of your users will suffer otherwise. The question isn’t “could we fill Dolores Park with people who want this feature?”, it’s “is this a valuable feature, within our purview, that all our customers will use?”.

9. But our competitors already have it

That doesn’t mean it’s a good idea. It could be something they’re trying out. It could be a shit idea. It could be something they’re planning on killing. It’s a mistake to assume that your competitors are in any way smarter or more tactical than you. Obsessing about your competitor’s features relegates you to permanently delivering “yesterday’s technology tomorrow”.

10. But if we don’t build it, someone else will

That doesn’t mean it should be in your product. If someone else builds it, do customers no longer need your product? Will they all switch over? Simply saying “someone else will” sounds good, but means nothing. We’ve all caught ourselves saying it. Often it’s the logic used to expand a product because you’re not willing to admit your product stops somewhere. You’re afraid to draw the line.

Here’s an example: A typical date might involve a movie, dinner, and a lift home. If a cinema owner is constantly worried about what other businesses will build, and hungry to capture more value, they’ll put a restaurant into their cinema and start a cab company. Then they’ll be weak at all three. Then restaurants start screening movies…

11. But the boss really wants it

If the boss is also the product manager, and has the necessary time and insight to make smart holistic decisions, then this is fine. However, if someone is trying to earn brownie points by focusing on pet projects that their manager has a penchant for, this inevitably leads to trouble.

12. But this could be “the one”

This is a classic “Appeal to the Unknown”. Editing a product requires some hard decisions about what to build. You can speculate that any unbuilt feature could transform your product. But speculation is all it is, nothing more. When you’re afraid to make hard decisions, you fall back on appealing to the unknown, and therefore building everything. You end up with a repository of features, not a product.

Why is “no” important?

The thing is, no one keeps crap ideas in their roadmap. Identifying and eliminating the bad ideas is the easy bit. Real product decisions aren’t easy. They require you to look at a proposal and say “This is a really great idea, I can see why our customers would like it. Well done. But we’re not going to build it. Instead, here’s what we’re doing.”

There are no small changes

Still not convinced that there are no small changes? Let’s play this out with a simple example.

“We want to limit the length of a text review in the product to 140 characters, because we may want to use SMS to send them out at some stage. That’s a small change, right?”

Wrong.

There are no small changes when you’re committed to delivering quality software. Let’s look at the above case. A naïve programmer may well get this coded in three minutes - after all it’s just an if-statement.

But product management is never that simple. Before you embark on any “small changes” to your product you need to ask some questions. Sticking to

our review length example from above, let's start with some easy ones.

What happens when the review is above 140 characters? Do we crop the string, or display an error message to the user? If we display an error, where does it appear? What does it say? Who is going to write the error message? How do we explain to the user why we're limiting them to 140 characters? How will these errors look? Do we have a style defined? If not, who is designing it?

But wait, there's more...

In the unlikely event that we have answers to hand for all of those initial concerns, we're still not finished. Just doing this server-side is a messy way to handle an error. We should do this client-side. But if we're going to do client-side validation then that throws up a few more questions...

Who's writing the JavaScript? Does the JavaScript display the same type of error as the server-side code? If not, what's the new style? How does it behave without JavaScript? How do we ensure that any future update to the 140 character requirement impacts both client-side and server-side validation?

We're still not done. Look at this from a user's point of view. They're already frustrated by having to limit a review to 140 characters for a bizarre reason they won't understand, and now we're asking them to guess how long their message is? There must be a better way. Let's give them a character counter. Oh, well that raises a few more questions...

Nearly there...

Who is going to write this character counter? If we're using one we found on the net, then who wants to test it in our target browsers (i.e. not just Chrome 37 and beyond)?

Also, where is the count of letters displayed on the screen? What does the count look like? Of course, the style should change as the user approaches zero characters, and should definitely look erroneous when they've used more than 140 characters—or should it stop accepting input at that point? If so, what happens when they paste something in? Should we let them edit it down, or alert them?

When we've implemented the character counter, styled all the errors, implemented the server-side validations, and checked it in all of our supported browsers then it's just a case of writing tests for it and then deploying it. Assuming your time to production is solid, this bit will be straightforward.

All of this happily ignores the fact that users will wonder why someone wrote an eighty word review just before them and now they're only allowed write a 140 character one. Obviously we'll need to keep support in the loop on this, and update our documentation, API, iPhone, and Android apps. Also what do we do with all the previous reviews? Should we crop them? Or leave them as is? Don't get me started on how we're gonna deal with all the emoji that people use these days… good luck sending them in a text message. We'll probably need to sanitize the input string of rogue characters, and this means new error messages, new server-side code… the list goes on.

Once you get through all of this you will have your feature in place, and this is just for a character count. Now try something that's more complex than an if-statement. There are no tiny features when you're doing things properly. This is why as a product manager you need a good understanding of what it takes to implement a feature before you nod your head and add it to the roadmap.

The big picture…

Often what seems like a two minute job can often turn into a two hour job when the bigger picture isn't considered. Features that seemed like "good value" at a two minute estimate are rightfully out of scope at two hours.

Key point: Scope grows in minutes, not months. Look after the minutes, and the months take care of themselves.

Agreeing to features is deceptively easy. Coding them rarely is. Maintaining them can be a nightmare. When you're striving for quality, there are no small changes.

Nothing comes for free

Often in software you'll hear that you get something for free e.g. "the framework just gives us this for free", or "if we use X service provider, we get Y for free", or your developers will tell you "we were building this bit anyway so we get this extra piece for free".

Nothing is free in software.

It's not free because you're sitting there talking about it, you're debating if you should do it. That's costing you time for starters. It's not free in the doing of it, or the verifying it was done right, that it works, that all edge cases are workable. QA is never free.

It's not free in the communication of doing it. If you add a feature, you have to tell your team, to tell customer support, to then tell your customers. You have to tell your customers, otherwise it's definitely pointless. Communication is never free.
It's not not free to carry it forward. Once it's there you'll have to support queries from customers, you'll have docs, how-tos, videos, all needing to be updated. It's not free to undo. Maintenance is never free.

Think of these features like house pets. Craigslist is full of free pets, yet you're not stocking up your house for a very simple reason. There's a big difference between the retail price and cost of ownership.

CHAPTER 3

Which new features to build

When it comes to product strategy at Intercom we find the Jobs-to-be-Done Framework is extremely useful in helping us to decide what we should build. Popularised by Harvard Business School professor Clay Christensen, it is a way of looking at the motivations of customers in using a product, rather than the traditional marketing techniques of slicing and dicing customers according to demographics.

People don't buy a product because they fall into a particular group e.g. female, mid 30s, living in suburbia, working part-time. But they do buy a product to solve a problem. If you understand the "job" that customers are hiring your product to do, then you can make sure that you have a razor sharp focus on helping them achieve the desired result. The features you choose to build should be the ones that will help them do the job that needs to be done.

Making things people want

The problems people encounter in their lives rarely change from generation to

generation. The products they hire to solve these problems change all the time.

When you're building or managing a new product, you have to believe you can create a better solution that people will want to use because it delivers a better outcome for them. A strong understanding of the outcome customers want, and how they currently get it, is essential for you to succeed in product management.

Maybe your customers want to be entertained, or spend more time with their friends, or understand what projects teammates are working on, or maybe they want to project growth for their business. If the desired outcome is real then they are already achieving it through some product in some way. Your job is to improve upon that.

Sidenote: If you can't find what product they're currently using, the chances are that it's a fictitious outcome ("Wouldn't it be cool if I could get all my social media files in one place?") or an aspirational one ("Of course I want to lose weight"). Espoused behavior never reflects reality.

Focusing on outcome, rather than category, industry, or product type, lets you understand your real competitors. The second a company focuses on "the industry it's in" rather than the "outcome it delivers", it loses touch, and shortly after, loses customers.

Newspapers, for example, believed they were in the "Newspaper Industry", and as such struggled to work out why bored commuters had stopped buying their product. They would look left and right at their competitors and wonder which newspaper had stolen their customers. They would experiment with new formats, new layouts, lower prices, sharper headlines, but they couldn't stop the rot. Had they instead focussed on the outcome they deliver (bored commuters want to be entertained for short bursts of time with bite-sized articles), then their competitors (Twitter, Facebook, news apps) wouldn't have been so oblique to them.

Let's look at some jobs that, like boredom during a commute, have stuck around for years, through hundreds of technological advances.

Age old jobs, new solutions

People, particularly students and young people, wanted to pass notes and messages, without fear of other people seeing them...
People still want this so today they use Snapchat.

People wanted to store photos in a safe place, like the shoe box under the spare bed...
People still want this so today they use Dropbox.

People wanted to put their favorite photos in a prominent place, like their mantelpiece, so everyone could see them...
People still want this so today they use Facebook.

People wanted to collect scrapbooks of ideas, for home renovations or other projects...
People still want this so today they use Pinterest.

People wanted to leave nice reviews, and tips for other travellers in physical guest books...
People still want this, so today they use Foursquare.

Doing a better job

There are literally hundreds of examples like the ones above and there's a common trend in all of them. Making things people want involves understanding a long standing human or business need and then using technology to:

- take out steps.
- make it possible for more people.
- make it possible in more situations.

The first approach, removing steps, is the most common for start-ups. Pick a need where the existing solutions are old, complex, and bloated, and find the simplest smallest set of steps possible to deliver the same outcome. Arranging a taxi in a city used to involve calling many numbers until you found a company with availability, then a lengthy dialogue about your location, destination and

required arrival time.

Today you press one button and a car shows up.

The second approach usually involves reducing the cost (in time or money), or barriers to using a product so that more people can use it, thus expanding the market. Fourteen years ago, if someone wanted to get their writing online they had to rent a Linux server, download a .tar.gz file containing the source code of a blogging engine, upload it, run a series of weird commands to unpack it and give it write access, and then configure it.

Today this is two clicks in Medium.

The third approach involves removing common situational limitations on a workflow. Accepting payment used to involve bulky machines with rolls of thermal paper, faxing paperwork to banks, ISDN lines, and batch transaction transfers run nightly.
Today you swipe a card through a phone and you're done.

Jeff Bezos is famous for saying "focus on the things that don't change". The problems that people and businesses encounter don't change often. The ways they can be solved changes almost yearly. So it stands to reason that making things people want should start with the "what people want" bit, and not the more tempting "things we can make".

Remember: It's easier to make things people want, than it is to make people want things.

An acid test for new features

Here's a simple set of Yes/No questions that you can quickly answer before you add another feature to your product roadmap.

In the previous chapter we wrote about how product strategy means saying no, but a list of reasons to reject a feature isn't as immediately useful as a test that any new feature must pass. So here's a list of questions your new feature must

score straight yes's on.

1. Does it fit your vision?

What do you believe that no one else does? What do you know about this problem that no one else does? How do you believe it should be solved? Anyone can pull the data, run the focus groups, read the Gartner reports, get out of the building, but only you have your vision.

Product decisions based on vision alone sometimes seem irrational, because they're tough decisions. Most people understand that their bug tracker doesn't also monitor server uptime. That's rarely debated, it's not a marginal call. But should a project management tool have a reporting feature? Not so clear.

The more nuanced decisions are the ones where you meet resistance. Colleagues, customers, even other product managers and founders will push back. Here's some examples:

- Apple refused to ship netbooks at a time when netbooks were the most popular style of PC. Every analyst demanded them.
- Basecamp refused to add Gantt Charts to their product. For that they were labelled blind ideologists.
- Developer Garrett Dimon refuses to add an "on-hold" state to his issue tracker, Sifter, as he simply doesn't believe it's the right way to manage issues.

Worse than being a hard decision, you'll never truly know if you got it right. There is no right. There is no wrong. This is art, not science. It's just you and your vision.

2. Will it still matter in 5 years?

It's a hard and boring question to ask but will this still deliver value in five years time? You'll feel like the curmudgeon of product planning. But that app that was so hot in 2013 was gone by 2014. Those slide, fade, and fold effects that everyone loved last year look tacky as hell this year. If you're going to spend the best years of your life on a product eschew the trendy,

and focus on the meaningful.

3. Will everyone benefit from it?

Beware the “fre-cently” bias. You never doubt the things you hear frequently or recently should be road-mapped. It’s a natural reaction caused by your inbuilt empathy for customers. It’s not nice to tell people “no” repeatedly and hear the same responses over and over, so when possible “fre-cently” is used as an excuse to reward yourself. “Sure we’ll build that, I’ve heard it twice today already,” says the founder with 4,800 daily active users, to the unbridled joy of 0.0625% of her customer base.

The danger of the fre-cently bias is that it gives you the illusion of analysis, the sense that you’ve done your homework and this is a rational conclusion you’ve come to. In reality you’ve made a lazy decision to satisfy the whims of a small sample of vocal users without taking a second to investigate how many people really want or need the feature.

4. Will it improve, complement or innovate on the existing workflow?

Adding a whole new workflow to your product should be a rare occurrence. The majority of your time should be invested in improving, complementing, or innovating upon existing ones, and for any given project you should know which of these you are doing.

If you’re improving the current solution, your metric will be customer satisfaction and/or maybe a decrease in tool time, or support requests.
If you’re complementing it, your metric will be an increased throughput for the workflow, as it now works in more cases or can be used in more circumstances.

Innovation is the trickiest. You’re shooting for the mythical “whole new way”, which carries so much risk but offers so much reward. Your measure will likely be new customers acquired though often they come indirectly, as a result of the PR, marketing or awareness created.

Redesigns are fun, but you can spin your wheels on them. A good way to cut through the bullshit is to simply ask “Will more people use it? Will people use it more? If neither, then will it definitely be better for those who do use it?”

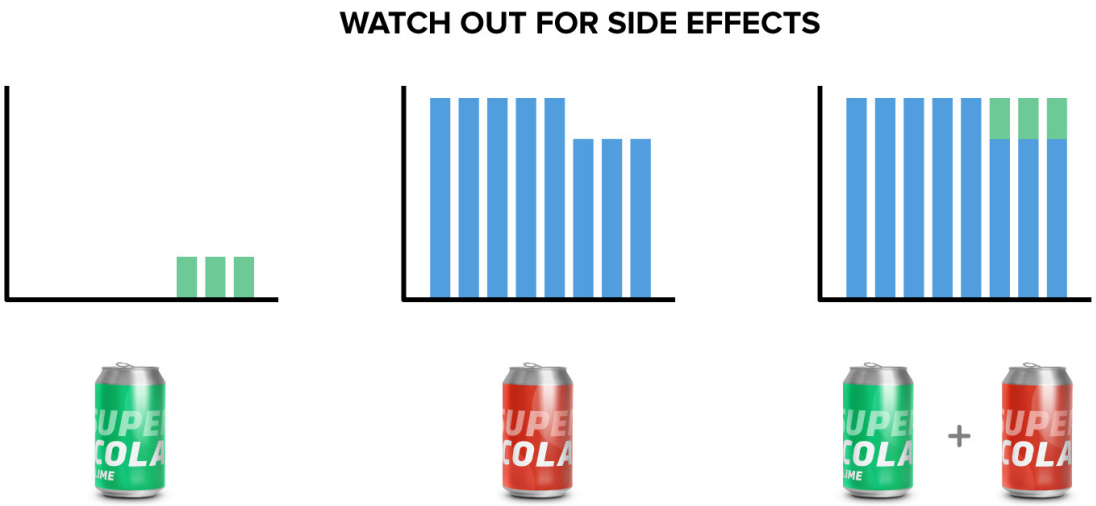
5. Does it grow the business?

Can you join the dots between the impact this new feature will have and new revenue? e.g. a project management software company might make the case for project templates, the argument being templates can be used in more cases, which should increase the number of projects customers have, which in turn increases upgrades, and thus revenue.

Note that reducing churn also grows the business; dollars don’t care where they come from. Often a feature is added to ensure stickiness of customers, or to widen the moat around a product. The key point in all cases is to understand how any work affects growth, after all everyone works on growth.

6. Will it generate new meaningful engagement?

Most metrics ignore the system and focus on the isolated feature being added. Put a new button in your product and people will click it. Get enough clicks and you can call that an increase in engagement. But that’s nonsense. A counter-metric is “have people stopped doing anything else?”. So if you add a metric to track one area of your product, you must also analyse the other areas that are likely to be impacted.



A real world metaphor is the effect of “Diet Coke with Lime” on Diet Coke sales. We see how launching a whole new workflow affects sales. If you’re the PM on Diet Coke with Lime, you could call your launch a success, but if you’re the CEO you’ll realise you complicated your product line, bought all these limes, all these juicers, and all that advertising, and have no new revenue to show for it. Not exactly a success, no matter what metrics that PM can show you.

7. If it succeeds, can we support and afford it?

One fallacy of “quick wins” and “easy hacks”, is they usually only evaluate effort required before shipping e.g. someone surprises you with a JavaScript bookmarklet for your app, or an agency produces a Windows Phone app in record time and low cost, and because the effort was relatively minimal, and the idea makes sense, you ship it. Success!

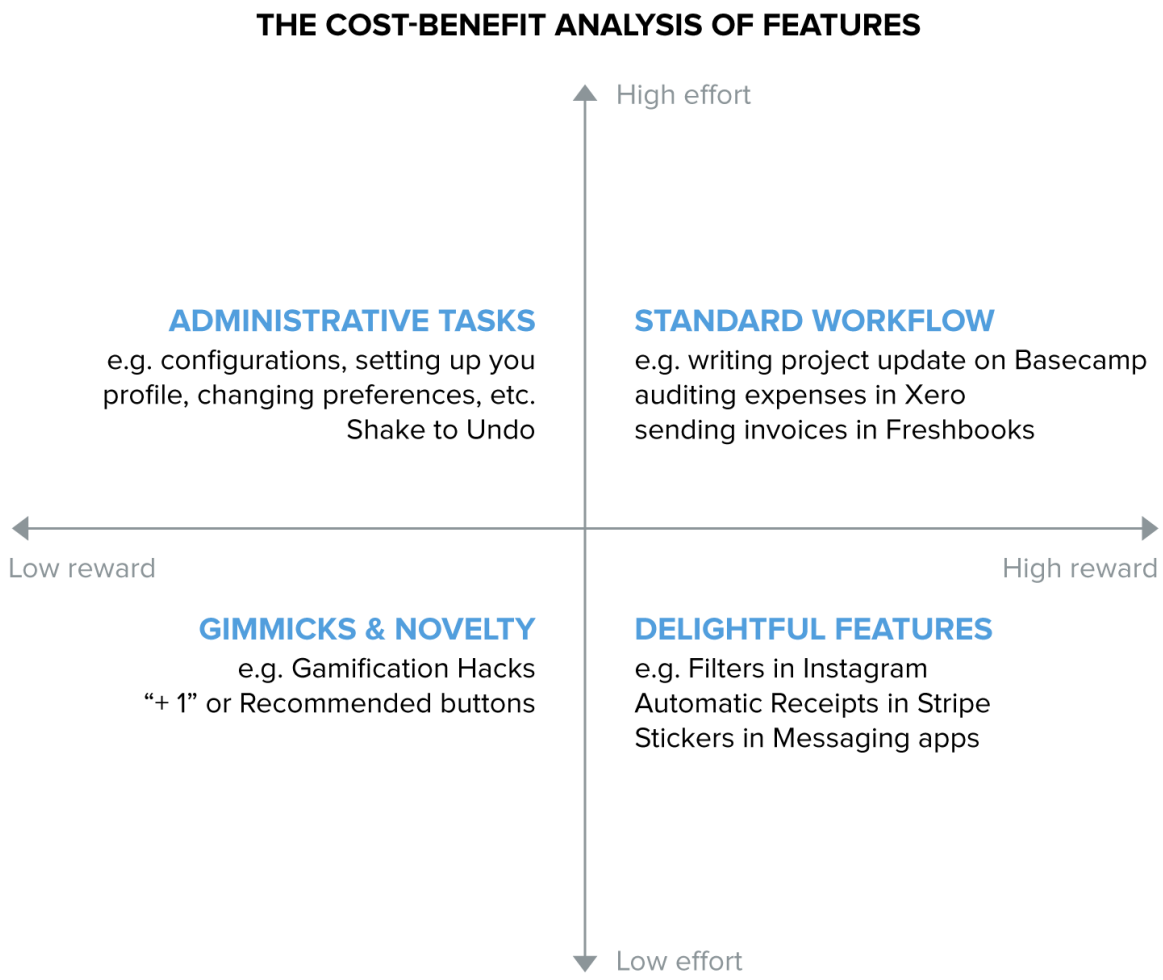
Then the support requests come in, and it turns out none of your team know XAML, so you’re stuck with a broken build live, and a few hundred customers complaining to support about it. Of course it’s never that obvious. Good engineers rarely say they don’t know something. They’ll hack at it some evenings, display some initial competency, and then estimate how long until they fix the bug. This is all time that you didn’t plan on spending when you shipped this app. And that estimate is wrong. And then some.

Another area that can bite is offering incentives, or initiatives that you can't justify. If you have a CMS product, and you offer free homepage designs, you'll get more sign-ups. It makes sense to do this early on, when you really need to entice customers to try an as yet unproven product, but it won't work long term. It goes without saying that doing things that don't scale is a great strategy, until you need to scale.

8. Can we design it so that reward is greater than effort?

As my colleague Paul Adams previously wrote, for any feature to be used the perceived benefit has to be greater than the perceived effort. End users understood the benefit Google Plus could bring, but the overhead of dragging and dropping many copies of your contacts into various boxes, on a regular basis, was simply not worth it. I feel that check-splitting apps habitually fall into this category. Splitting a check can be a pain point, but any solution seen thus far still costs too much, and we're not talking about price. We're talking about time, overhead, social capital, etc.

Product design is about cost-benefit analysis. How useful is something vs how hard is it to do. Here's how I think of it...



It's important to know which quadrant you're building in and if you're wise you'll steer clear of the upper left for all but the most essential of features.

9. Can we do it well?

Every product has its neglected corners, usually areas where the PM dabbled but conceded defeat. Conceding defeat all too rarely results in removing a feature, it usually just means ignoring it, leaving a messy trail and a confused offering.

The problem here is when product teams tackle areas they don't fully understand. This is often the case when a team moves beyond self-design, that is, past the point where they are their customer. At this point their design process breaks, and they need a new one. Examples include:

- teams that don't track time adding a time-tracking feature.
- people who don't manage calendars designing a calendar management option.
- designers who don't close issues building an issue tracking system.

Note that none of the above are inherently wrong, what's wrong is the design process. It needs to move beyond "works fine for me" into something much more activity focussed. To build a feature well, you have to understand the job it does intimately. As a corollary to the old saying: if you can't do it well, it ain't worth doing.

10. Can we scope it well?

Starting with the cupcake release for a feature is essential. Early usable releases provide the feedback necessary for an idea to flourish. A good sign that a feature isn't well scoped is when it lacks specifics, e.g. "Make it work for bigger companies", or that it's feature based e.g. "reports", as opposed to job-based e.g. "Let sales managers see their team performance".

It's always easy to agree on truisms in product meetings. Someone says "It should be easier for bigger teams", everyone nods, and a Post-it gets stuck on a whiteboard. It's only in the specifics that you'll understand the scope e.g. "we should let people group their colleagues by department" vs "we just need an enterprise mode".

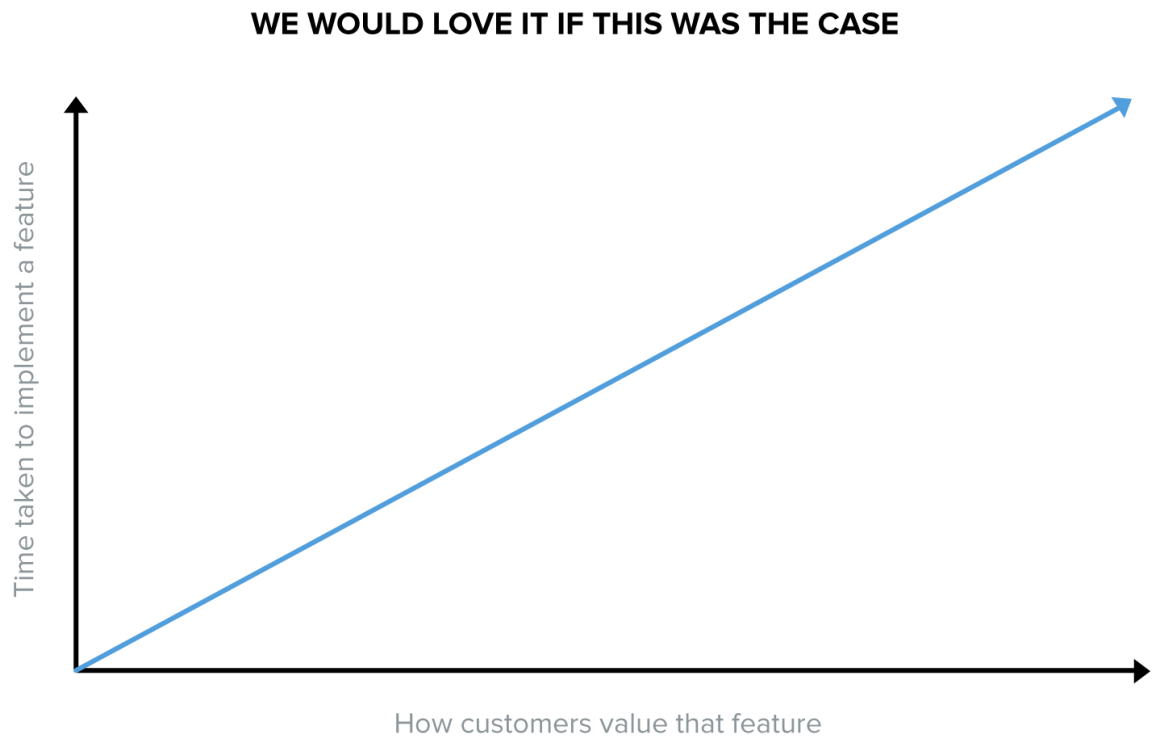
Badly scoped features meet no ones requirements, ship late, if at all, and add nothing to the product but confusion.

Slippery slopes & marginal calls

It's tempting to excuse occasional violations of this list, assuming that "as long as it's right most of the time", it'll be fine. Maybe that's true in theory, but this is software. Reality bats last here. This is why we say that product strategy means saying no. Roadmaps are incredibly hard and require agonising trade-offs, but regardless, every good product manager needs a firm checklist for when they say yes, and when they say no. And they don't make exceptions.

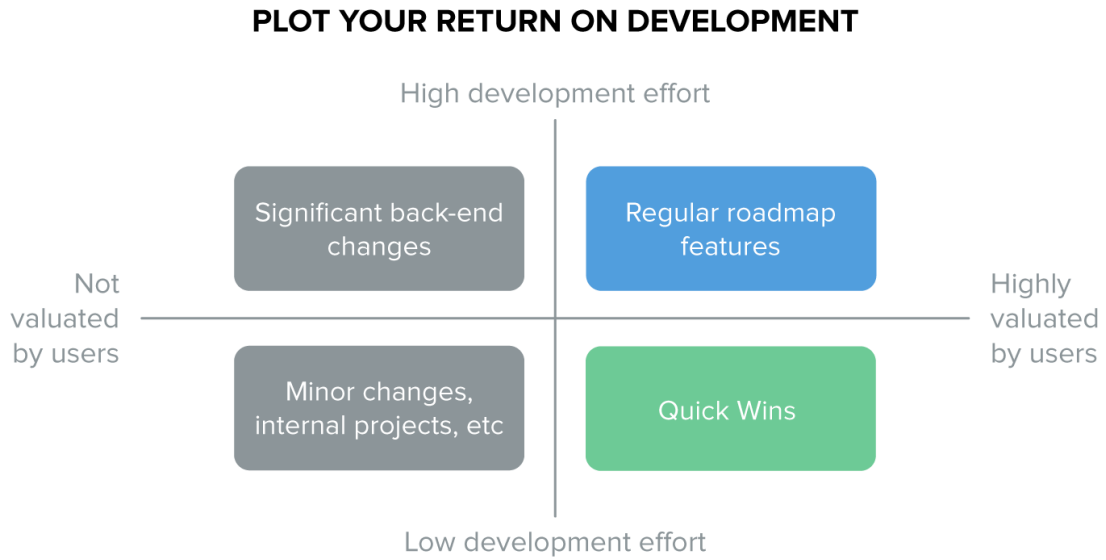
Products get bloated one lazy decision at a time. There is no single choice you'll make where some light turns on saying "Your Product is Now Too Complex", and you'll never get an email saying "Last Thursday at 2:03pm Your Product was Disrupted". Bloat, complexity, and disruption can only be seen in the rear view mirror, so always be mindful of the risk of saying yes.

Features & physics envy



Physics envy is a term that can be used to describe the desire of designers, developers, and product managers to enforce laws on a system that simply doesn't obey them. We would love it if the return on investment was always perfectly proportional to the time spent. It would make product planning easy. The problem is that you can spend five weeks working on a killer feature, only to see it go ignored by your users. Conversely you can add one sound effect, have your cute logo say "Hi", or make your favicon blink, and all of a sudden everyone is talking about you. It's not rational, but then what is?

Rather than try to stare enviously at those who don't work with such irrationality, let's see how we can use this to our benefit. When you're planning your next few months work, plot your features on this chart...



Anything in the lower right corner is special. These are features where the return is disproportionate to the effort. Some examples I've seen are:

- rewrite the app copy to make it more personal, funny, precise or useful.
- add a couple of keyboard shortcuts to let power users be more productive on the main screen.
- remember the last active project and bring the user back there when they next log in.
- provide extra information and links to actions in email notifications.

When to use quick wins

Customers won't value all the development you do on a project. Some necessary tasks such as back-ups, re-factoring, optimization, improving infrastructure offer little immediate reward. This doesn't mean you can ignore those issues. The trick is to plan your roadmap so there's never long periods where customers feel the application has been abandoned.

This is precisely where quick wins are useful. You can schedule these quick wins to cover time periods where it would otherwise appear that nothing is happening. This way you're constantly delivering value to your customers, whilst still tackling the larger features or back end problems. Remember the only thing worse than an app in constant flux, is one where users can see cobwebs forming.

Rolling out new features

Roll outs are make or break and they're damn hard to get right. Doubly so if you're changing your user's workflows. For a lot of business to business products, it's someone's job to use it. There are thousands of people whose entire job is to use Intercom. So when we are considering changing a behavior, or adding a new one, we don't do it lightly.

We follow a series of steps to progressively release it using feature flags. Your mileage may vary, but we've had good success with this approach.

Team testing

The first step is to deploy it live with real data, but only visible to the team that built it. This is important because it avoids either too many people reporting the same issues and also permits a release before every single step is perfect. The team themselves know how far they've gone, so they alone can tell what's a bug versus what's simply incomplete.

Company testing

When a team believes their feature is complete, it is released it to the entire company who immediately use it in their workflow. There are no elaborate explanations, tutorials, or excuses. It's simply shipped, just like we intend to ship it to customers. This catches any confusions we might have created; if life-long Intercomrades don't understand it, what chance do our customers have?

Restricted beta

The first public release is to a single digit percentage of users. We maintain a "Trusted Testers" group for this purpose. We communicate it as a beta, and specifically ask for feedback later, once they have used it. That final point is subtle, but it's worth remembering. Feedback on what it's like to use a feature can only come from people who have used it. Not people who tested it or played around with it. Jake Knapp wrote that "Reactions > Feedback", citing how when people are trying to be helpful they'll make up things in an effort to offer you improvements. Speculation and espoused behavior have no place in product design.

What you're looking for here is:

- **Discoverability** - are people finding this feature?
- **Engagement** - are people using this feature?
- **Adoption** - is it now being used as part of a workflow?
- **Use Cases** - how is it being used? what use-cases are popular?
- **Barriers** - Who isn't using it? why? what's preventing them?

During restricted beta you can ensure it's discoverable for all users, refine your marketing to focus on pitching the true value as your customers see it, and you can resolve barriers to usage and adoption.

Full roll out

This is when it's available to everyone who is eligible for it and you need to think about how you are going to tell them. Typically not all features are available to all users - there's some basic price discrimination at play - so at a minimum you split your message into the "haves" and "have-nots".

Those eligible for the feature should be sold on it. Feature announcements are often very inward looking, focusing much more on "what we did", rather than "what you can do". If you want people to use your product, encourage them by showing them what they can use it to achieve.

Those ineligible should be sold on the feature as a reason to upgrade. Often you

can achieve this with a free trial (e.g. Try our premium plan for one month free).

Message schedule for the feature

Once a feature is launched it's easy to forget about it and move on to the fun of designing the next one. But features without engagement are the seeds of bloatware. For any significant feature, you need to have plans for those who use it, and those who don't.

When people are using it well, you want to gather use cases, and marketing collateral from them for future use. Depending on your plans you may also want feedback.

When people aren't using it, you want to encourage them, educate them, and engage them, the only acceptable outcome here is they'll start using it, or they'll tell you why not.

There is no pride in having a "big" product with lots of features, only a highly engaged one. The more surface area your product has, the more important it is that you chase engagement and learn from your mistakes. This is most important when you're young and scrappy. You're capable of shipping a new feature every week, but you should focus that energy on growing usage of your product, not growing your product.

CHAPTER 4

Getting that feature used


JUST
PRESS
FLY!

Deciding what features you need to build and getting them built is really only the start of the process. New features are just code that's gathering virtual dust unless they are being used by your customers. As discussed in Chapter 1, if a feature is not being used it's time to make hard decisions about killing it or making it better. Fail to make a decision one way or another and you risk losing customers to a nimbler competitor with a more focused product that does the job more efficiently.

Launching a successful feature demands the same skills as launching a successful product. The difference is that you also have to navigate around all of your legacy decisions, and appease current customers too. It's tricky.

The majority of new features flop. You just don't notice it happening. And that's the point. "New improvements" sit unappreciated, unused, and are eventually cast aside. Welcome to software. Improvement is hard.

You do months of research. You pull demographics, metrics, ethnographics, analytics, psychographics, you name it. You meet with customers for days, weeks, months. You meet their parents, you feed their dogs, you feed their parent's dogs. You do it all to understand what customers really need. Not what they say they need. Not what they think they want, or what they're asking for. What they really need. And you go and build that!

And it still flops. This is a tough racket.

Why new features usually flop

The thing is, unless customers use a feature it may as well not exist. This is often forgotten in the rush to ship fast. It's not just about shipping code to servers, or checking boxes on a roadmap, it's about getting your software used. So before you ship that next feature, ask yourself these questions...

1. Will everyone see and understand it?

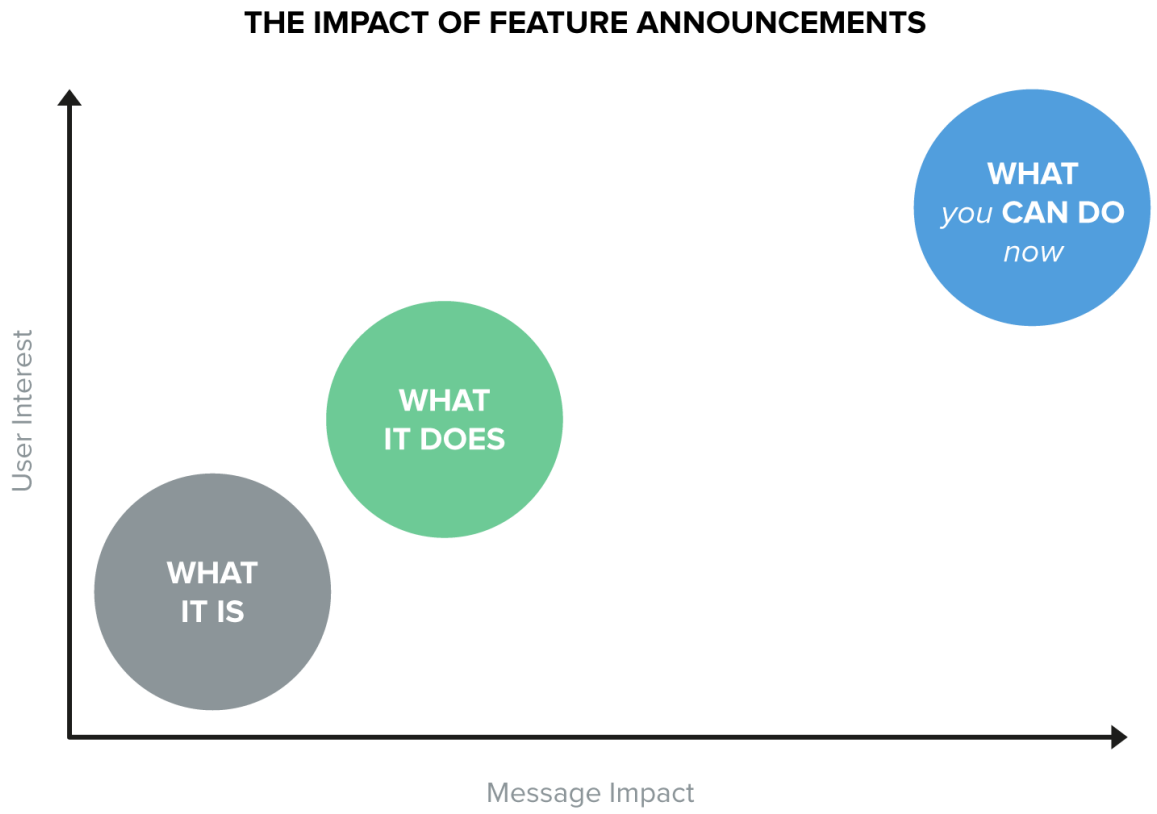
Fun fact: when Microsoft asked their users what they wanted added to Office,

they found 90% of the requested features were already there. Microsoft assumed this was an awareness challenge, hence the launch of the ribbon toolbar which highlights all features, and therefore highlights nothing. Like I said, this is tricky.

When you design the first version of a product, you want it to look complete. So you don't necessarily plan for expansion, or leave space for future features. As you add them you make space for them, hastily, causing discoverability problems for your users. If you find yourself constantly fielding questions about where a feature is, then you have a discoverability problem and all new features will flop, or at least require lots of training to find and adopt.

2. Are you showing users what you did, or what they can do?

Telling your customers something is a "ground up rewrite", "HTML5 based", "responsive" or anything like that will miss the mark unless you're selling to developers. No one cares what you did, or often even how you did it. Your customers care about what they can do.



Focus your message on what your users can now achieve with this feature and you'll get their attention.

3. Are you announcing it in context?

New features, especially small additions, land in products without much context. Your goal should never be to "just get it launched". The goal should be to "just get it used". Email is the wrong medium for these announcements: it's often overkill, and usually arrives at the wrong time and in the wrong context. The right time to promote an improvement is when someone is in your product in a position to use it i.e. an in-app announcement that is triggered according to rules and events.

4. How will tomorrow's sign ups hear about it?

As a product grows, it expands beyond what's learnable in one sitting. That's not a problem, not every feature makes sense up front. Features are only rele-

vant when they solve problems. Put simply you don't need to tag files until they are uploaded. You don't need to merge tags until you've got lots of different ones. So it stands to reason that telling new users how to merge tags is a badly timed message.

There's a right time to introduce features to customers, and this is where targeted messaging matters. We promote saved replies in Intercom once a few replies have been sent. We promote keyboard shortcuts after a user has used the product long enough to care about them. This is how you make messages matter.

5. Do you plan to follow up with users & non-users?

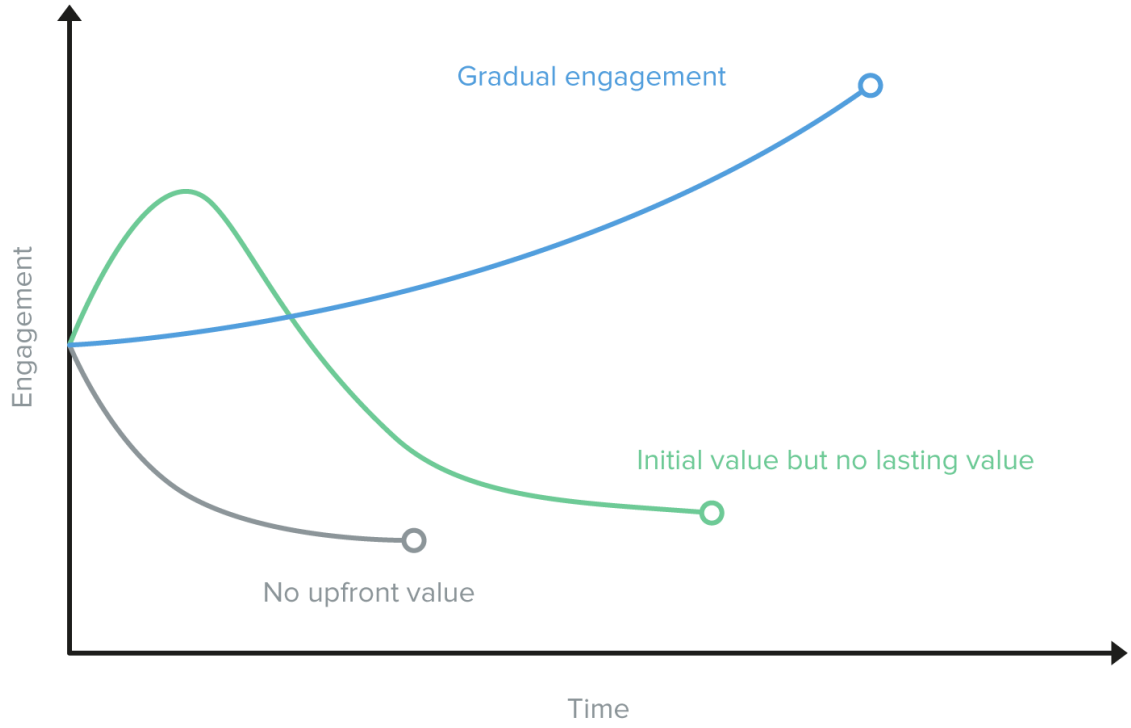
Once a feature is live, you should follow up with those who use it to understand how, why, and when it's used. Look for ways to increase usage of it. Here's some questions I've asked customers about a new feature:

- When did you notice it? What did you think? Did you use it straight away? What attracted you to it?
- Did you need to read the documentation? Was it sufficient?
- Were there any barriers to you using it?
- Are there times in your work day that you want to use this but can't?
- Have you told anyone about what you use it for? What did you say?

It's equally, if not more, important to follow up with those who don't use it and understand what's stopping them. Often you'll find barriers that are easy to break down e.g. "I keep forgetting to check back on it", "I don't know if anyone else in the company uses it", "I need to be able to get a CSV of the data", etc. These are all resolvable problems once you understand them.

Your design process must acknowledge that you'll get things wrong. So many roadmaps and sprints ignore this. Designers and product people get things wrong all the time and when they do, they're faced with two choices: improve it, or ignore it and jump onto the next feature. Do the latter enough times and you too will be telling the world that 90% of your feature requests were already in your app. Hope you like ribbons!

Increase user engagement



The definition of an “engaged user” varies from product to product. For a to-do app an engaged user should be logging in every day to add and complete items whereas for an invoicing app an engaged user might only log in once per month. There is no consistent quantifiable definition of engagement across different products.

Unlike page views, visitors, returning visitors, and conversions, there’s also no analytics app that can instantly tell you what you need to know. But ignore engagement at your peril.

Google+ claims 170,000,000 users, which gets them a few news stories, but ignores a very real problem. Almost none of their users are engaged. They’re just people who clicked a link titled “OK” when faced with Google+. It’s similar to newspapers goosing up their page views using hacks like slideshows or multi-page articles. At some point you have to wonder who you’re really fooling.

Engagement is just one piece of the puzzle, but it is so frequently ignored that there's lots of quick wins to be had. Here's three ways to increase user engagement.

1. Make a strong first impression

Every day a potential customer is seeing your interface for the very first time. This can be forgotten within teams that are always designing for themselves. The first screen your users see has three important jobs...

- Explain how your application works.
- Motivate your users to get started.
- Let your users know how to get help, if and when they need it.

Web applications that do none of the above get the exact behavior that they designed for. The users will log out and most likely never return.

There are lots of great ways to welcome a user to your app and show them around. There are blank slates, tutorials, dummy data, etc. A good first step that we encourage Intercom customers to do is to post a welcome message to greet each new sign up personally. This works great for starting conversations, which in turn increases conversions.

Simply communicating with your users is a great way to encourage them to ask questions, try out features, and stick around. By starting a dialogue they're far more likely to say things like "How can I email inactive users" or "How should I use tags?".

At best you'll win yourself more customers, and at worst you learn what's missing or misunderstood in your application.

2. Gradually expose the depth of your product

Every worthwhile application has features that aren't immediately apparent or useful. These can include quick-wins such as email notifications and alerts, third-party integrations, export features, and even small optimizations like keyboard shortcuts.

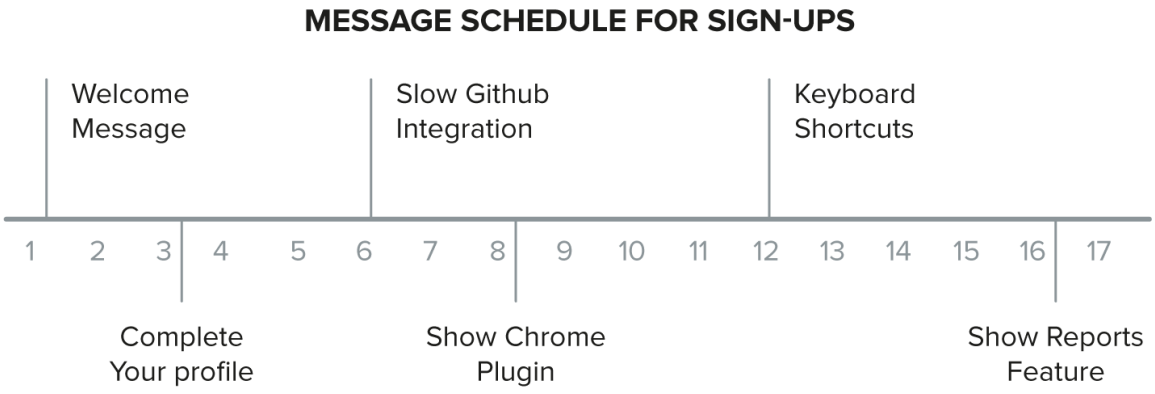
These deeper benefits can't be called out immediately. After all, who cares about data export before they have data, or keyboard shortcuts before they've used any features?

Most product owners tend to try to expose these features either through badly timed email blasts, documentation, or an FAQ. None of these approaches work well.
Emails arrive out of context, out of time, and are more likely to interrupt a user than to interest them. Their response is to archive your message and stop reading future emails.

Leaving features to be explained and promoted in your FAQ or help sections means their only chance of exposure is when a user has experienced an issue with a part of your product. This isn't the ideal time to distract them with other features.

We advise our customers to create message schedules to gradually promote certain features. When you have good insight into your userbase you can work out which secondary features delight users and at what point they're useful. Once you've done that it's just a matter of timely communication. Intercom lets you stagger messages over a series of sessions. This means each time a user logs in you're showing them more and more reasons to stick around.

Here's an example engagement routine from one of our customers:



Always end each message by letting your customers know that you're there to help if they have any questions. This is key to getting feedback, which helps you tweak your engagement routine.

3. Announce features and improvements in-app

Users don't notice when your product development slows down. They're logging in to use your product, not monitor your development progress. However, if things go quiet for long enough, they'll be easily distracted when a competitor releases a new feature, whether valuable or frivolous.

The only time your users care about features or improvements is in your application, so that's exactly where you should be announcing them.

We see a 10x increase in communications (as do our customers) from in-app messages over email announcements, and there are other, softer, benefits too. For example when we announced the new map sharing features in Intercom, our users clicked straight through, were immediately impressed and started sharing the location of their customers around the world. I can't imagine an email achieving a similar effect.

Develop your message schedule

Scenario: You're a product manager who wants to get your customers engaging with your product more often, when they're outside the office. To achieve this you've developed a mobile version of your app. Now you need to tell your customers about it.

The zero points method here is a blanket email shot, to all registered users, right now, with a link to the mobile app. What happens next?

Some customers will receive it while at their desk. Some receive it while using the website on their phone. Some will receive it who haven't used your product in years. Some get it at 4AM, others at 10AM.

Your reporting tool tells you that you had a 4.43% click-through rate and you'll be disappointed as this was a huge stressful effort. The worst response is to say "5% clicked? That's great, Just sent 19 more of those mails and we're golden."

Can't we do better than this?

Alternative approaches

You could split the customers into two groups: those who've previously logged in to the product from their phone, and those who haven't. That would let you target a better message to each group.

You could create a permanent message in the app for anyone using it on a mobile device so they know what they're missing. You could email everyone after they next log out, so you know that they've recently used the app, and it will be fresh in their minds. You could write a hilarious message, poking fun at competitors, and include a funny graphic. That'll get people talking which helps spread the word.

You could take a lesson from Slack who make their release notes in the App Store funny and enjoyable to read. e.g.

"Fixed: We've fixed the loophole in the last release that allowed Gif to be pronouced "Jif". As you were.
Fixed: When the keyboard was dismissed by swiping down, the chat window would then partially scroll back up, which was interesting, but not in a good way."

All of these are options that go beyond the default "Let's just mail everyone loads" approach that most companies pick.

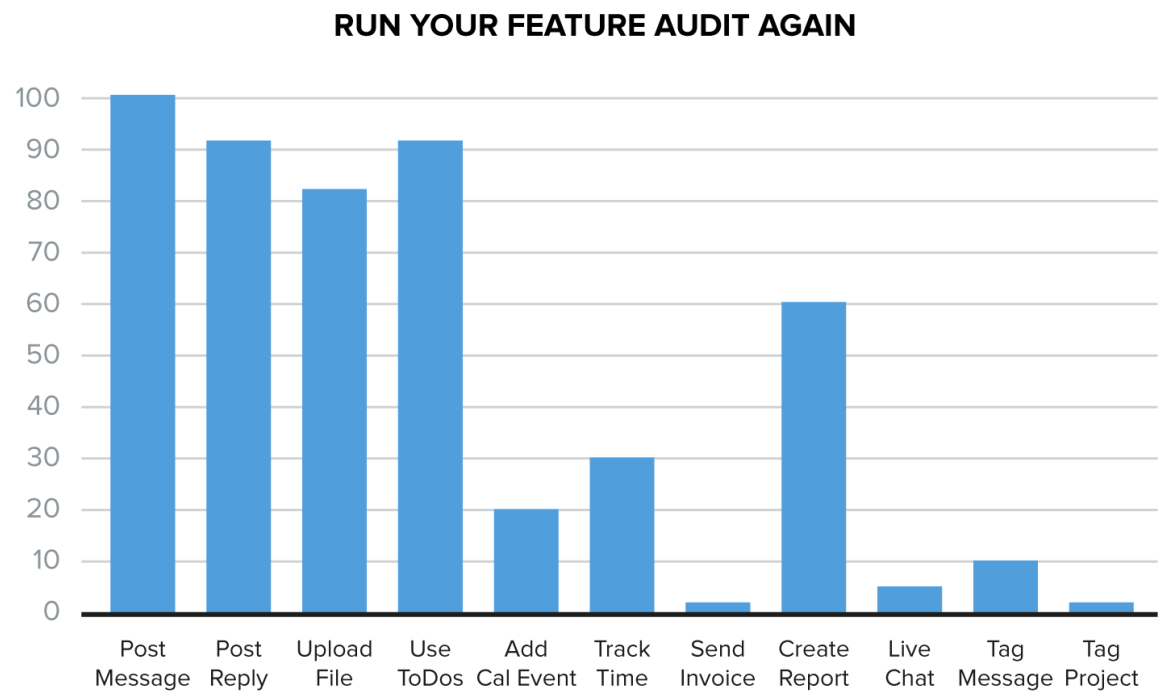
Be comfortable killing a feature

When you think about feature creep and bloated products what comes to mind? Endless tabs, toolbars, settings, and preferences, right?

For a five year period you couldn't sit through a talk on UX or product strategy without seeing some variation of the infamous Microsoft Word screenshot, with all the toolbar options turned on, leaving a tiny space at the bottom of the screen for the user to type anything. It's the perfect example of the need for simplicity.

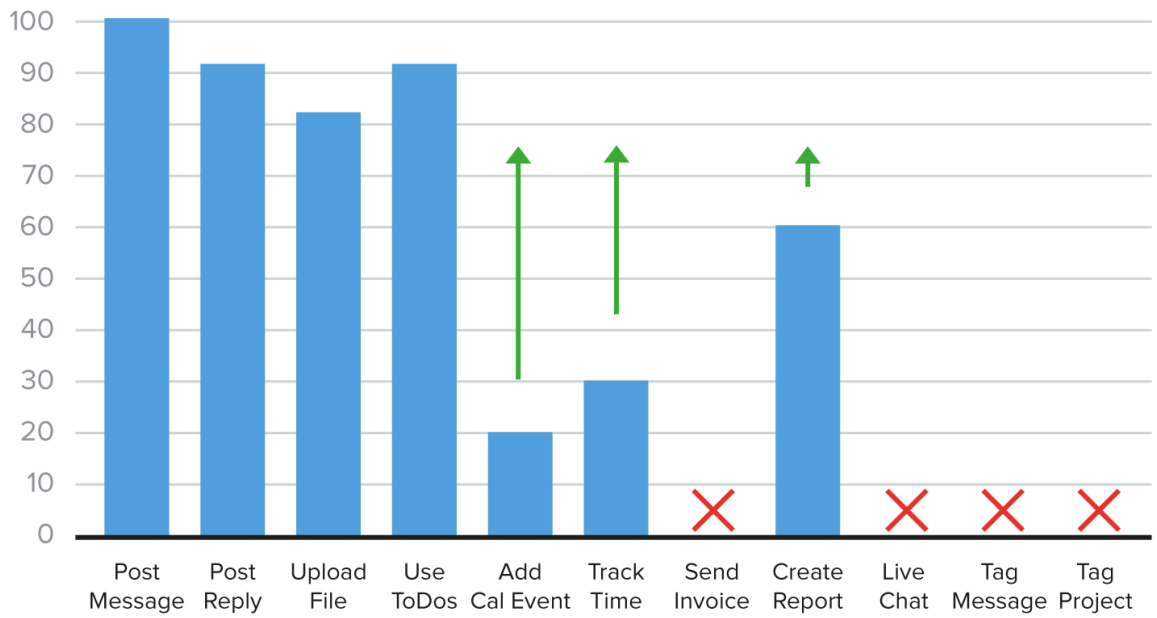
Poor user interface like that highlights the cost of feature creep, which is why you need to be comfortable killing a feature. There is simply too much stuff for users to comprehend. The problem is that horrible screens like that Microsoft Word one don't inform the product manager about what to do next. Sure there's too much shit on the screen, but what do you do about it?

To solve feature creep you need to identify which features are being adopted by everyone, and which ones are struggling to gain traction. It's time again to run the feature audit, explained in Chapter 1.



When you have any type of long tail of features that have little usage, you've gone off course, or your strategy is "build everything that everyone wants". The latter might sound crazy, but products like Microsoft Word have very done well with that plan. But you're probably not building Microsoft Word. If you want a well defined product, you must be comfortable killing features.

Fish or cut bait



Features struggle for adoption for lots of reasons, users don't see the value in using it, they find it too complicated, it's hidden somewhere clever in your product, they're the wrong type of users, and dozens more. This is why it's important to say no. When you're faced with a feature that only 8% of your user base are using, you have to make a call: Kill it or Keep it.

Killing a feature can take many forms, you can hide it from new users, message current users to explain your decision and give them time to prepare. Or you can just tear it out of the UI and never mention it again. Both work, with varying amounts of overhead. The bigger the product i.e. the more users you have, the less you get away with dramatic moves.

Keeping a feature that has no adoption requires design and communication. You need to target the users who are using it, and work out why. Target some users who aren't, and work out why. Obviously this is where a tool like Intercom shines as you can target the users and start the conversations in seconds.

Often you'll find simply promoting the feature through in-app messaging works. We've worked with customers who have driven up to 30% increases in

feature adoption simply by messaging the right type of users at the right point in their lifecycle. Similarly we've seen customers get enough raw feedback from customers to make it clear that a feature simply isn't valued, which makes its removal painless.

Good product owners let in very few dud features. Great ones kill them on sight.

Conclusion: A work in progress

We had to make difficult calls about what points and pieces to include in this compilation. We tried to zero in on the posts that would be useful for someone either early in their career or who has recently transitioned to product management. Our thinking on how to run, manage and create products is a work in progress, and this is our first stab at compiling it.

If you found this useful check out our blog Inside Intercom where we regularly post our thoughts on product management and related topics such as design, the business of startups and customer success.

We don't have all the answers, but we're growing a healthy list of questions.

Thank you for reading this book

We'd love if you shared your newfound wisdom with friends:

Facebook | Twitter | Linkedin