# the no bullshit guide to product management

SHAHID HUSSAIN

# Contents

# About this book

This book is about product management for organisations building software, explained as simply as possible. This is a short book, because it isn't very complicated.

*"If I had more time I would have written a shorter letter." - Benjamin Franklin*

## Who is it for?

- You're already in or about to start a product manager role building software - something on the web, or for a mobile or embedded device. If you work in consumer goods or any other industry, I recommend a different book.
- You don't have the time or patience for a fat textbook.
- You cringe at words like synergy, vision and "action" as a verb.

## What do product managers do?

Product Managers plan out what to build, help build it, then check that it worked. They also keep everyone pointing in the same direction.

# Before you build your product

## Figure out what to build

You probably wanted to work on online or mobile products because you enjoy using them so much, and have some sense of what makes them good or bad. Now that you're in the driving seat, you might be tempted to just start making stuff. Don't. Before you begin, you should understand clearly not just what you are doing, but why.

First, you need to know what your organisation fundamentally wants. Do you want to make money right now? Scale the business? Or just make the world a better place? Most organisations want to do all these things. The question is, which one do you want to do first? You should figure this out way before you start building products.

- You might want more profit in the short term. You can get this by increasing what your customers pay you, or increasing the number of paying customers.
- You might want to scale the business by investing in something, in the belief that you can make money back later on.
- You might want to squeeze cash out of a successful product to invest in something new.
- You might want to cut costs to survive in a tough economy.
- You might just be a good egg that wants to help people in some way.

Let's say that your fundamental goal is to scale the business. Step by step, consider what you can do to make that happen.

- The best way to scale the business is by adding users.
- The best way to add users is by helping users tell their friends about us.
- The best way to help users tell their friends about us is to add a social media sharing button.

At each step, ask yourself - this is one way, but is it the best way? Figuring this out is entirely dependent on the circumstances, but you might consider - which one will have the most effect on the users or revenue? Which one are you good at? Which one will cost least to implement? Which one might give you a unique advantage over the competition?

Here's the same list, including alternatives.

- We could scale the business by increasing revenue per user, but the best way is to add users.
- We could add users by increasing traffic from search, but the best way is to help users tell their friends about us.
- We could help users tell their friends about us by paying them a referral bonus to do so, but the best way is to add a social media sharing button.

This process will help you identify the best way to attack your goal.

## Figure out what to build next

You have a good idea now what to do for product one, but consider that:

- You will probably be building more than one product at once.
- You should be planning ahead for your next few products.

The good news is that you will never be short of products to build. You are a resource, your team is a resource, engineering is a resource - and the worst thing you can be doing is nothing. On top of that, other departments will and should be bugging you to build product X or Y which will help them do their job better - a bug tracking system for engineering, an ad product for sales, and so on. The next product might be one of these - for example, if bugs aren't getting squashed effectively because there's no way to manage the bug queue, fixing this will help users more than changing a button colour. In short, the next product should be next most effective at attacking the fundamental goal.

When you're planning out your list of things to do, there are two common traps to watch out for.

- Building the next shiny product is tempting. Once a product is launched and live, it looks like you're done, but you're not. You know that, but it will never be obvious to everyone else - so there will be pressure for you to move onto the next thing. Resist. Iterating on your product, once live, is as important as creating the product itself - so ensure you're building in enough time for this.
- Jumping around between types of product can be inefficient. If you're working on a social feature right now, jumping to a mobile app will be tougher than building another social feature next - you'll need to get your head out of one place into another, your engineers may need to learn a new technology, and so on. This is dependent on your people - some like hopping around, some hate it. Your engineers might have institutional knowledge about a new area, or they might not. Make sure you're being efficient about switching product types.

How far ahead should you plan? Generally, the larger your organisation, the more you should plan ahead. Larger organisations have more people - more people means more opinions and more folk you need to convince.

# Figure out what to kill

Killing products is just as important as making them. Like pruning a plant, cutting off dying parts makes the rest stronger.

It is tempting to just leave old products running, simply because they're already built and appear to cost little or nothing to run. However, consider that:

- More features means an overall product that's harder to use - so removing unused features gives you benefit for little cost.
- As the product grows, keeping feature X running when you introduce products Y and Z can take resource if there's a conflict - so even though you aren't spending time directly on old feature X, you are increasing the amount of time taken to build Y and Z.

In the same way you figure out which products to build, you need to figure out which products to kill.

- Which features aren't being used by our users?
- Which features aren't giving anything valuable back to us? (e.g. money, reputation, warm fuzzy feelings)
- Which features are costing us the most to run? (e.g. maintenance, operations issues)

Killing a feature is hard - you are exchanging a piece of functionality in the short term for making a product easy to use in the long term. Change is hard for everyone - especially users. Your most involved users are used to the way things are, and are the most vocal when things change - some

of them may even leave. If that is the case, you are making a bet that you'll acquire more or more valuable users than the ones you lose.

# Building your product

You are building a product for your users, so before you do anything at all, you must understand them.

Then, you can start the build process:

1. Have the general idea of what you want to build in your head.
2. Figure out how users interact with it and generate some wireframes.
3. Figure out what it looks like - colours, fonts, and so on.
4. Create a document (called a specification) that defines the product end to end.
5. Create a plan for release.
6. Ask Engineering to build it, and answer questions as they come up.
7. Release the product according to the release plan, and observe data and reactions.

## 0. Know your users

Ultimately, you are building a product which will benefit your organisation. You can build with this in mind (Is This Good for the COMPANY?), but when it comes to the nitty gritty of layouts, colours and flows, it leads to better products when you focus on the user.

First, you must understand who the users are. Grouping together set of users who share common features will help you. This is what business types call "segmentation". You can segment based on a whole bunch of parameters, and the choice is very dependent on the product and your intuition of the market. To start you off, here are some common ones.

Demographics, like age, location, education level, income, or occupation.
Psychographics, like social class, lifestyle, or personality type.
Behaviour, like how heavy a user they are, what they use the product for, or how much they love your organisation.
Think carefully about the groups of users you can identify. Surf your user forums, interview people, observe the interactions on your website. Do some people write comments while others rate items? Some younger users know your product inside and out, while older ones are confused by it? Some use it at home, others use it at work?

Next, you can build personas. Personas are little caricatures of users who represent segments that you think are important. For example, if you built a mobile cooking app that some people use professionally, you can invent a user called "Raymond the Chef" to represent those professional users. Create a back-story and a personality that is typical for the group. This will help you build a product with Raymond in mind, and help others in the organisation understand why a product is being built that way. As you see interactions from users or speak to them directly, you should be able to identify them by persona (e.g. "he's a Raymond") and fill out your understanding of the personalities you are creating.

Creating segments and personas is the best way to attack the problem with larger products, but it's a lot of work. The easy way is to design for yourself - this is what some call "scratching your own itch". If you are bang in the target segment, you get instant feedback (from yourself) as a product comes to life - hence products can come together very quickly. Keep in mind that, no matter how typical you think you are, you're

probably way more technical and know way more about the history of a product than any user.

However you do it - always keep in mind the user as the product gets built. It's amazingly easy for smart people to spend weeks designing and building an amazing new feature, without realising that nobody had a problem with the old one. This will help:

Do user testing and watch the results with everyone in the department / company on a weekly basis. It doesn't have to be in-depth or even something you conduct yourself - just something quick, done online. You can focus testing tasks on whatever you are working on at the moment. Forward all customer feedback to everyone in the team. Bear in mind that your users aren't always right, but if you believe they are giving you the right signals, this is an amazing and cheap way to go.

# 1. The general idea

Staring at a blank whiteboard is scary. What IS the idea? How DO you improve the signup process? How CAN you stop users leaving the site at step 3? Don't panic - there is a way to get started.

It's both tempting and possible to go straight to the answer. You might be smart or experienced enough to go straight to the best solution, and if you need to solve something quickly, this is the only choice. If you have a bigger problem and it's most important to try to get it right, try following this process, which flatcap-toting designers call "ideation".

First - go wide and pull out all the sites, apps, products that made you say "that's awesome" or "my five year old could have pooped that out". Don't restrict yourself. Ask your colleagues. Ask your pals. Ask the user

community. Most importantly, don't tell anyone that their idea is dumb - dumb ideas often spark off better ones. People are smart, and having lots of perspectives will help you see the problem from every angle.

Then, narrow down the ideas to get the most sensible. Good ideas are usually ones that solve a problem for a user (these are sometimes called pain points.) If you know users are having real trouble logging in, or want to communicate with each other but can't, these are problems that need solving. The evidence for this is what's happening right now on your site, on other sites where your users are going, through interviews, and so on. Listen closely to your users, but don't take everything as gospel - to quote Henry Ford, "If I had asked people what they wanted, they would have said 'faster horses'."

You are using your judgement to pick the best ideas - so is everyone else. Check to see if they tally. If they don't, debate it - defend your ideas and ask them to do the same. In the end, the call is yours - you are ultimately responsible for the success or failure of this product.

You don't have to agree about everything. Some of the best products come out of one person defying everyone else and building what seems crazy. Many of the worst products happen that way too. You have the choice - allowing crazy things to happen means you are taking more of a risk for more potential reward. Whichever way you go, avoid groupthink, which is compromising every sensible idea just to make people happy. What's most important is user happiness.

Once you have settled on an idea - make it solid. If the goal is to add users, make sure you're starting to track the number of users right now, so you have a baseline to compare against later.

# 2. Wireframes

Wireframes are pictures of the product that are just lines and text - no colours, no fancy styles. Wireframes are there to define what happens when a user clicks X, drags Y or pinches Z. (The process is called interaction design.)

You and your designer will work to construct a set of layouts, at each step deciding what the right user interface element is. Should you use horizontal or vertical tabs? How much contextual information is necessary? Swipe or click interaction? What happens when the device rotates? If you spend a lot of time surfing sites or apps, you probably already have a good idea of which ones you liked using - go back, and figure which interactions that seemed natural. (Remember that the best tools are ones that you don't notice.)

Now that you have a vague idea of what the product looks like and what happens when you click X or Y, you can start to test it.

One way is paper prototyping. Print out all of the wireframe screens you've created, sit a user in front of them, get them to tell you what they think, and point to where they would click on the page if it was real. When they point, swap out the paper with the next screen. An alternative is to put every screen into keynote / powerpoint, create links over the buttons and interactions and link to the appropriate next screen. Either way, you can get reactions from potential users very early in the process, and make any changes way before it gets coded. The earlier you can improve your product, the better.

# 3. Colours, fonts and so on

This is where the wireframes are used to generate a full mockup of the site or application, so you can see exactly what it looks like before it's built. It's often called visual design. If you're building really fast, or your team can code the front end faster than they can photoshop, skip this step.

If this product is part of a bigger site, your fonts, colours, button styles and so on may have been defined up front. Stick as closely as you can to your style guidelines, or the standard widget set - it's faster and creates consistency. Consistency is hugely important to your users.

During this process, keep in mind that how your product looks is more than just aesthetics. Having a beautiful product sets up an expectation of quality.

# 4. Create a specification

A specification is a document that defines what the product looks like and how it behaves in every circumstance. You should aim for completeness, but recognise that it's hard to get there (especially for a complex product).

Engineering will say "I'm done!" once the specification is complete. It's the goal they're aiming for.
After code is complete, the product will be tested. Those tests are derived directly from the specification.
The actual process of documenting the product will help you think through edge cases.

Edge cases are situations that can occur, but aren't things you initially considered. For example - let's say that you create a user preferences screen. What does it look like if someone goes to this screen when logged out? Define some behaviour for this case and use it to fill out your specification.

# 5. Create a plan for release

As the specification is a bible for engineering, the release plan is a bible for everyone else as the product is released. There are lots of different ways to release a product, and the larger your user community is, the greater risk, and the more careful you should be. For example, one way to mitigate risk is release a product internally (called "dogfooding"), or to a limited set of beta testers to get feedback before releasing it to all users.

The release plan is a list of who is doing what, on what day. The plan is a list of dates, activities and the person responsible. Here's a sample plan:

12th May: Initial communication to beta group (Bob)
13th May: Release to beta group (Alice), begin collection of feedback (Bob, Me)
17th May: Code freeze (Alice)
18th May: Release to testing systems (Alice)
19th May: Communicate to wider user community (Bob)
24th May: Release to live systems (Alice), gather feedback (Bob, Me)

# 6. Ask engineering to build it

While the product is being built, you should have done enough work up front to feel like you can post the specification off to the developers and go work on the next thing. The reality is that your engineers will absolutely need to talk to you - when making a project real, every detail of the specification becomes important. If you've missed anything, they will let you know, and every occurrence means a potential change in functionality.

At the same time your engineers are keeping you on your toes, you should be doing the same. You probably have assigned a limited time and number of engineers to work on the product. Make sure that the development is keeping to schedule. If it isn't, you have a choice - do you cut features, or extend development? Rushing to cut features to meet a development deadline can easily lead to a substandard product. On the other hand, extending development can lead to projects that never complete, and development will always extend to fill the time allotted.

# 7. Testing code

Starting a project is fun. It's ideas, it's something new and shiny. Anyone can start a project, but this is the hard part - completing it is the real test.

As the product nears completion, your engineers will (or should) start running standard tests to verify that it meets the specification. For web applications, there are a few copies of the software on different machines - engineers shouldn't be working on the same system that users are on.

- The development system, affectionately nicknamed "dev". This is where engineers are actively creating new stuff.
- The QA (Quality Assurance) system. This should be an exact copy of the live system. (Usually the QA is still slightly different than live, simply because it's difficult to simulate the load of hundreds of thousands of users, or a caching system.) If you are testing really quickly, or your site is still small, this system might not exist yet.
- The live system (sometimes called "prod" or production), which users are using. Contains stable, tested code.

You should also be playing with the freshly minted product on the development system - does it feel like what you expected? If there are serious problems, or if you have a brilliant new idea, you have the option to stop, rework the specification and ask your engineers to make changes. If it feels good, it's time to move forward to the next step.

# 8. Testing on users

The product is completed and working, and you're itching to release it to everyone. Before you do that, you can release it to some users and see how they react. There are a few ways to do this.

Release an alpha / beta. Find a subsection of your users who are happy to try new products, give them your new creation, and ask for feedback. You get free appreciation from your most loyal users, anticipation from users who don't have it yet, and you get to mitigate the risks of releasing to everyone at once. On the other hand, it's not as quick.
A/B testing. Here's an example of what that is. Let's say your product is a redesigned form to let users sign up. You can easily set up your system so it shows half of your visitors the old form, and half the new form - then check if more visitors who got the new form signed up. If you get lots of

visitors, you can just show the new page to a small percentage of visitors, minimising the risk to your traffic while still keeping the test valid.

## 9. Release the product

As a product gets released, someone should be communicating with users carefully about what's happening. Tell the user community up front what is coming and why you're doing it. You may worry that they'll hate it, and that's possible, but it's certainly best you know that up front, and most of your users will appreciate being part of the discussion.

You've created a product you're happy with and communicated a plan to everyone for the release. Ideally, you should just follow the plan, and watch carefully for user reactions and usage data from the product.

That concludes one release cycle.

# After your product is built

Your product is done, launched, out there in the market, and people are using it. Congratulations, you are about half done.

When you came up with the initial ideas for your product, you basically guessed. It was an educated guess - you looked at data, talked to people, used your experience and so on - but you couldn't say with absolute certainty whether it would work, because you've never launched exactly this product into exactly this market before. Now that you have launched, you have a golden opportunity to adjust the product. It's like playing crazy golf - you whacked the ball vaguely in the right direction, but now it's stopped closer to the hole, you have a better chance with each putt to sink it and pelvic thrust at your friends in victory.

You should now have two important types of information coming at you.

- Statistical data from the product
- Personal reactions of your users

Both of these are massively important. Data will tell you what is happening, people will tell you why. These are yin and yang - without both, your understanding will be incomplete.

# Statistical data

If you're building a digital product, you should be able to reap masses of data from it. There's no reason not to use data gathering tools on every part of the experience. First, a set of very basic statistics, like pageviews, uniques, and clicks on any interface element. Then, a set of more interesting statistics that are harder to get, but will tell you more.

- Track batches of users through the product flow. Where are they dropping off?
- Get demographic data on your users, identify which ones are important to you (e.g. perhaps older users are more likely to purchase), then focus your analysis on their behaviour.
- Next, you need to pick out what's important from this data. Go back to your list of goals - if you were trying to increase traffic from search, examine whether this did or didn't happen. The data may pinpoint an issue - for example, if you've created a five step process, and you're losing 70% of users at step three, that step needs examining.

# Personal reactions

Once you have a good idea of what's happening, you need to know why. If you are losing users at one stage of the flow, it might be because your copy is confusing, the link to the next step is hard to find, or the system was too slow for the user. Until you know the answer to why, you cannot fix the problem.

To do this, you need to find and observe test users. Ideally these should be the segment of users you care about - users that are just like the persona or personas you created this product for. This can be hard, but

you can get closer to find them with highly targeted ads on sites like Facebook. Want to find people of a certain age, education level who like product X in a certain area? You can find them and offer them something to talk to you, or pop in for an hour while you observe them doing a task with your product. Careful with your incentive - you'll need to set it high enough to get them to bite, but not so high that it distorts their feedback. (Check the resources section for an alternative approach.)

# Pulling them together

Once you have an idea what is happening and why, you can solve the problems and release an updated version of your product. Now, you can start the process again - analyse data and personal reactions to see where the new problems are and fix them. As you continue to iterate, two things will happen:

Your product will get better at attacking the goal. Iteration will improve your product, but bear in mind that you can only improve what you have built already. If the initial premise was flawed, you should remove the solution and start again.. In crazy golf terms, if you were shooting at the wrong hole in the first place, putting closer to it won't help.
The benefit you get for each iteration will decrease. There's a lot of work involved, and while the first few iterations may yield great benefits, the fifth or sixth iteration may yield little.
Therefore, you must decide at what point you are done, and move on to the next task in your list. Others will pressure you to move on as quickly as possible, because a product looks done once it's launched - stay firm, understand the value of iteration, and move on when you believe you are ready.

# Your Team

Your team could be just you, you and an engineer, or a whole department. In a smaller team, one person will be doing multiple roles (especially you, mister product manager). In a bigger team, everyone can specialise, but you will work hard to coordinate everyone - so either way, it's important that you understand what everyone does and why.

## Engineers

Engineers are the tip of the spear. You will point the team in the right direction, but it's engineering who will make it happen - whether they do it well or poorly will show clearly in the product's quality.

Like Skittles, engineers come in different flavours. For the web:

- Front end. They create the presentation of information - colours, layout and so on. For the web, their skills will likely be in HTML, CSS, Javascript and similar technologies.
- Back end. They create the engines that calculate the information itself - a search algorithm, for example.
- Operations. They keep your servers passing data to hungry browsers around the world. If your site is small, you might do this yourself. If it's

large, you'll need someone with a deep knowledge of servers, clustering, routers, and so on.

For a mobile application:

- App developers, who wrote code specifically for a mobile operating system like IOS or Android. This is usually most of the work.
- Back end work is needed for more complex applications, like an image recognition app. In this case the app would take a picture, upload it to the back end for processing, and return the results to the user.
- Operations people aren't as commonly needed, because the mobile device is doing a lot of the work. However, if your app has a back end, you will need operations people to look after those servers.

Good coders are deep thinkers and problem solvers. They love working on a defined project, getting to a known goal (which is often "Eureka! It works!") and they absolutely will appreciate your direction and support.

Engineers will focus on creating an elegant technical solution. You need to balance this with thinking deeply about how your users see it. Sometimes a fantastic product has messy code behind it - it'll work, but be difficult to maintain and scale. Sometimes a terrible product has elegant code behind it, and it doesn't matter if it's easy to maintain or not.

# Project Managers

Project managers are organisers. They make sure that everyone is communicating, that the project is going to be delivered on time, and nudge along anyone who needs it. They know that a smooth process means that multiple projects can be handled at once, and they can predict

more accurately when a project will be completed. There can be a degree of overlap with your role - both of you have a direct incentive to make sure that development happens smoothly. However, their role is often focused on organising the engineering team, and they only start on a project once the specification is completed.

Project managers want to have a smooth process, allowing them to more easily predict completion dates and plan everyone's time. That's not always possible - sometimes creating the best product possible means having a degree of chaos in the process.

## Community Managers

For products on the web, the user community is massively important. Consider that customers in an area like retail don't actually talk to each other much - it's not often that someone in the queue next to you in Target gives you their opinion on the kitchenware section. The web is different - your users are talking to each other all the time. Your organisation should be part of that conversation - that's what community managers do.

Communities often follow the "90 9 1" model - 90 percent of your users will be readers, observers, consumers - 9 percent will talk or produce content now and then, and 1 percent will talk all the time, and heavily influence the behaviour of others. The most powerful influence on consumers is word of mouth, and this is where it comes from online.

Community managers want to get products that satisfy the existing user community. You need to balance this with the needs of future users you hope to attract with this product.

# Analysts

Analysts are similar to engineers in that they work with complex systems and large amounts of data. However, they aren't as focused on solving problems - they examine data to see what kind of behaviour it suggests. That allows you to do two things:

Optimise the kind of behaviour that you want from your users. For example, if you are guiding them towards a purchase, you can see at what step they are dropping out.
Identify behaviour that you didn't expect. There are only a few people inside the company thinking about how a product could be used, while there might be hundreds of thousands of users outside the company thinking about the same thing - and some smart alec discovers something you weren't planning on. Data helps you spot it, and either prevent it (if it's harmful) or encourage it (if it's wonderful).
Analysts will spend a bunch of time on research. Here's what that actually is.

- Site analytics: This data tells you exactly what's happening on your site. To cover the basics, you should be tracking all pageviews or screen views, and every click on every object on the site (no surprise to discover this is called "clicktracking"). If you have the time or resources, you should also be tracking visitors across the site, allowing you to understand what the most common paths are from page to page.
- Primary research: The analyst finds a bunch of users and asks them questions, like "How many times a day do you use our site?" It's very easy for this to be biased - for example, this question assumes the site is used more than once a day.
- Secondary research: The analyst reads a study that someone else has done and figures out how applicable they are to the immediate situation. This is where Forrester, Gartner, eMarketer and friends are

used. For example, if a Forrester study reports that US smartphone users are using check-in services more, that might indicate a feature for your product.

Data can easily be misused in an organisation. Primary and secondary research is not a measure of what is happening, but a guess at what might happen, and assumptions are always necessary. Furthermore, because deep analysis is time consuming, the results often become gospel - even more so when backed up by words like Forrester and Gartner. If you're analysing, be sure that your methods are open and easy to understand. If you're examining the results of an analysis, understand the assumptions and the methodology - these can easily flip the results of a study on their head.

As a product manager, you probably have some keen insight in to the behaviour of people as they interact with technology. It's possible to draw conclusions from what you know, and skip all of this time consuming research stuff. That's absolutely the right thing to do if you are making a non-critical decision very quickly. If not, be aware of two things - one, your competition probably have the same insights you do, and two, you are looking for surprises in the data, and you don't know yet what might be in there.

Analysing data is like unwrapping a present. If it looks like a book and doesn't rattle when you shake it, it's probably a book - but a surprise makes the best present.

# Design

Designers are half problem solvers, half artists. They are constantly solving the problem of how to make something easy to use. They are constantly striving to create something beautiful.

Making something easy to use is hard. Add functionality, and it becomes harder to use. One solution is to present the appropriate functions at the appropriate time. An excellent example is In 'n' Out burger, a chain of fast food restaurants in California. As you walk in, there's a menu pinned above the counter that's incredibly simple and familiar - cheeseburger, hamburger, coke, shake, fries. No confusion. However, there is a "secret menu" containing variations of the defaults. This is something users often discover after a few visits - just at the point where they needed a new option. No boredom. Continue to add more depth as a user's experience grows, and you'll keep them coming back for more. An excellent example of this is World of Warcraft - as you progress, you discover more about the story, your character, and can do more in the game. However, I wouldn't recommend that you try it - because of this balance, it's highly addictive.

Making an application where users notice how amazing the design is should not be a goal. The best tools are ones that you don't notice - your screwdriver has a grip, your tea cup has a handle, your shoes don't hurt your feet. Everything that doesn't have an impact on how usable a site is - colours, styles, fonts and so on - is really marketing. These choices come together to give a site a feel - and that feel helps define the emotions people have when they use your application. Does it make them feel businesslike, or welcome, or edgy? That's the definition of brand.

Designers want to create a beautiful product they can be proud of. You need to balance this with getting your product out early so you can test a reaction, and correct as necessary.

## Marketing and Strategy

Marketing and strategy people often have posh suits and MBAs, and an important function. They consider whether a product is a good idea in the first place, and if it is a good idea, how to make them feel good about buying it. That means two things - brand and strategy.

- Brand is not a logo or an advert - it is how people feel about you.
- Strategy is not a huge, sleep-inducing powerpoint presentation - it is the answer to the question: "Why will this work?"

Marketing's job is to improve this emotional reaction, because a user's feeling about a product rules how much they would pay, whether they would use your service again, and so on. If someone says "Walmart" do they think about cheap prices or mom and pop stores going out of business? If someone says "Apple" do they think about style or pointless expense? Emotion is defined by everything from a celebrity endorsement to the font in the logo.

Marketing and strategy people use a couple of tools to make sure they've covered all the obvious questions. The three Cs: Customer (who are they), Competition and Company (understanding our own capabilities). The five Ps: Product, Price, Promotion, Place, Partners. For each of these aspects, consider the options. How much should I sell my app for? Would I sell more if I lowered the price? How much more? Is it worth the tradeoff? Should I promote the app? Would the expected bump in sales be worth the cost of promotion?

Marketing and strategy focus on cash flow and keeping the organisation afloat. You need to balance this with doing what's best for the user, so you can create a great product for the long term and provide a fair exchange - they get something good, you get something good.

# You Yourself

Organisations all work differently, but there's often one role that pulls everything together. At tech companies, that role is often the product manager. That's you.

Unfortunately, that doesn't mean that you get a cushy office, afternoons on the golf course, a secretary and your people talking to their people. As a PM, your team probably doesn't report directly to you - you are there to serve the team, not the other way around. Here's how to do that:

- Make sure everyone is communicating.
- Point everyone in the same direction.
- Do the boring work that no-one else wants to do.

## Communication

The most important skill you have in this area is the ability to listen. In other words, the ability to shut up, understand where people are coming from and adapt your approach to get the best results. Sometimes that means changing what you're saying - remember that people talk about the things that are important to them. Sometimes that means changing the way you're saying it - the best way to do that is by copying the style of the person you're talking to.

- In their speech, loud / quiet, or a fast / slow talker.
- In their body language, relaxed and laid back or leaning forward and intense.
- In their choice of words, technical or simple.
- In their demeanour, confident or reserved.

Don't make it wild and obvious - just check yourself if you spot that you're talking all brash and confident to someone who isn't, or vice versa. This puts you both on the same level, and you'll find that you suddenly understand one another much better.

People in a team need to communicate. Some hate talking to anyone else, and want to just get on with real work. Some want to know exactly what everyone else is doing and give their opinions. Most people communicate less than they need to, but broadcasting everything to everyone (especially in a large company) can also be a waste of time.

Getting the team talking usually involves a meeting. These can be incredibly useful or an incredible waste of time. Here's how to make them work:

- Meetings make decisions. Make those decisions firm and restate them at the end of the meeting, so there's no confusion.
- Force meetings to start on time and end on time. If anyone is late, start anyway.
- Keep track of time during the meeting. Kill the conversation quickly if it's getting irrelevant.
- Inviting people who don't need to be there is just covering your ass. Don't do it - it wastes everyone's time, including yours.

Passionate people will argue out their opinions. Those opinions are valuable. Don't argue all the time - it's exhausting, and can drive the team apart. Most importantly, don't agree all the time - that leads to a lack of

clarity. It comes right down to the very words you use in a meeting, or a document, or an email. The more you compromise to avoid conflict, the less clear anyone is on anything. Here's how that language can change:

- X is the best option
- X is probably the best option
- X may be a good option
- X, Y and Z are possible options
- The choice of option is probably due to some other aspects of the issue, A and B, which are still unknown, and we need to progress our understanding on these to a more satisfactory place. It would be best to get some leadership on this from department P owing to their deep expertise in area Q. Feel free to follow up on this either with me or department P, and after the appropriate process has been applied, we will look to follow up with some thoughts on X versus Y. I would love to be as engaged as we can be going forward, so please keep us in the loop. Regards, an asshole.

Suddenly, you won't be able to make any gutsy decisions and progress in the company will slow to a crawl. Everyone will start bitching about it at the pub after work, pinning the blame on someone else for how it went wrong. That's a company no one wants to work at - don't let it happen.

## Point everyone in the same direction

Everyone in your team has a job to do that contributes to part of the whole, but that whole looks different from everyone's perspective. When you're in the thick of it, it's absolutely natural to focus on your own work, and that's usually a good thing. Sometimes it can be damaging - your engineers want to create a technically manageable solution, your analysts want something trackable, your community managers want something

that won't ruffle feathers, and so on. Your job is to keep reminding people that they're all shooting for the same goal. Focus on the user. No-one wants to create a product that sucks.

Passion is what drives good people to do good work. Your job as a product manager is to get people passionate, and pointing in the same direction. Part of that comes from your passion. If you truly believe this is the best product for your organisation to create right now, it rubs off on others. If you don't believe it, nobody else will.

## The boring work

Everyone in the team has a specialised skill to contribute. Your specialised skill is that you have no specialised skill - you are a jack of all trades. That means you can fill in the easy, boring stuff where needed. If some ongoing analytics need updating, do that. If a status report to someone needs writing, do that. If your engineer's email has gone down and needs fixing, do that. Everyone should be doing stuff that no-one else can do - except you. You're the mortar that makes bricks stick together. The grease that makes the wheels turn.

## Finally

As a product manager, you have to power to fix any problem. It's your responsibility. You set the goal at the start, you encouraged everyone to chase it. You'll believe in what you do. If you can do that, you can create something amazing for your users and your organisation.

That's changing the world, a piece at a time.

# Resources

## Wireframes

If you're looking for wireframing tools, try these:

- On the web: Balsamiq, Mockingbird.
- On your computer: Omnigraffle (not free), Illustrator (not free), Inkscape (free).
- On your desk: A sharpie and a piece of paper.

Remember that what's important is the output, not the tool you used.

## User testing

The best option for user testing is to see users face to face, and understand their reactions. That non-verbal communication - a momentary smile, a frown, a raised eyebrow - will speak volumes about the emotions of the user while they use your product, and this is exactly the information you need.

Alternatively, there are a bunch of sites you can use to do user testing. None are as good as seeing someone's expression change with your own eyes, but they are quick, cheap and absolutely better than nothing.

- usertesting.com
- userlytics.com
- loop11.com

# Thanks

Firstly, thank YOU for reading this book - I hope you enjoyed it. If you didn't enjoy it, put this down, and go have dinner with someone interesting. Life's too short to do anything dull. If you've read this far, I'd love to hear from you - you'll find me at shahid@shahidhussain.com or on twitter.com/shahidhussain.

Books like this are a collection of experiences, and I've been lucky to have some great teachers. Thank you to everyone who I've worked with and learned from.

Angie Shelton, Danny Horn, Sean Colombo, Gil Penchina, Pamela Ramali, Erik Nordby, many many professors at Kellogg (especially Harry Kraemer, Kent Grayson and Don Norman), Katina Johnson, and all my other colleagues and friends.

Thank you also to everyone who helped me bump version numbers on this text: Sean Colombo, Pamela Ramali, Shaykat Chaudhuri and Sanjeev Kalia.

Now quit reading this, go out there and build something!