Chapter 5: PRODUCT MANAGEMENT VS. ENGINEERING

Building The Right Product Versus Building The Product Right

If a great product is the result of combining a real customer need with a solution that's just now becoming possible, then it's easy to see why the relationship between the product manager and the engineering team is so critical.

The product manager is responsible for defining the solution, but the engineering team knows best what's possible, and they must ultimately deliver that solution. As a product manager, you'll quickly learn that if you have a good relationship with engineering, then the job can be a great one. If you don't, let's just say that you're in for some very long and frustrating days.

One key to this relationship is for each of you to understand that you are peers—neither position is subordinate to the other. As product manager, you are responsible for defining the right product, and your engineering counterpart is responsible for building the product right. You need both. You need to let your engineering counterpart do what he believes necessary to build a quality product, and he needs to give you the room to come up with a valuable and usable product.

Both sides can be a huge help to each other. Specifically, the engineers can be a big help to you as you work to discover a winning product. Remember that they generally know what's possible better than anyone else.

Here are three ways you can use your engineers to come up with a better product:

1. Get your engineers in front of users and customers. Not only will they learn a great deal from seeing users struggle first hand, but they will get a better appreciation for the issues and their severity. This is often the inspiration for much better ideas and solutions. You can jumpstart this easily by inviting an engineer along to your prototype testing.
2. Enlist the help of your engineers in exploring what's becoming possible as technology develops. Brainstorm the different technologies that are available or coming available, and how they might help solve the problems at hand.
3. Involve your engineers (or at least a lead engineer) or architect from the very beginning of the product discovery process to get very early assessments of relative costs of the different ideas, and to help identify better solutions. One of the most common mistakes that product managers make is to come up with a great product definition, and then throw it over the wall to engineering. That just postpones the critical negotiation process of what's wanted vs. what's possible until there's not enough time to be able to make good and informed decisions.

Similarly, you can actually be a big help to your engineers. Here are three ways you can help them do their job:

1. Keep the focus on *minimal* product. More on this later, but your job as product manager is not to define the ultimate product, it's to define the smallest possible product that will meet your goals. This point alone will fundamentally

improve the dynamic between product management and engineering.

2. Do everything you can to minimize *churn* once engineering begins to develop the product. Churn is changing requirements and product definition. Some churn will be unavoidable, and engineers understand that some things are beyond your control, but remember that this is not the time for trying out your latest-and-greatest ideas.

3. There will inevitably be questions that arise during implementation, including use cases that were missed or weren't completely thought through. This is normal, even in the best of product teams. However, your mission as the product manager during the implementation phase is to jump on their questions and get answers as fast as humanly possible, always keeping the focus on minimal product and minimizing churn.

With all this in mind, I always try to encourage the best engineers to come try their hand at product management. I remind them that it doesn't matter how great the engineering is if the team is not given something worthwhile to build. And I point out the many great products and companies that have been created by an engineer who knew what was possible and tackled the bigger question of what to build. It's great for their career development, and can be great for the product (and therefore the customers and the company).

How Do We Succeed With Remote Developers?

One of the most common situations today is where the product manager is in one location, and the engineering team is somewhere else. I don't only mean outsourcing to India, either. The remote development team might emerge from an acquisition or merger, or possibly your organization is large enough where the developers are located in a facility you are not.

When the developers are not sitting right next to you, then all of the normal challenges of communication and execution are magnified, often to the point where many teams get extremely frustrated with remote development, and some members openly challenge the benefits of the purported cost savings.

If you find yourself with a remote development team, there are three key things you can do to dramatically improve the odds of your success:

1. The farther away the team is from you—and the more the communication challenges of language, culture and time zones—the more pressure there is to ensure that you have done a very thorough job on the product spec. The nightmare project is where the product manager isn't sure what he wants built (or keeps changing his mind) and the remote engineering team is thrashing. It's like being on the wrong end of a whip—and a recipe for failure.

In the chapter *Reinventing the Product Spec*, I write about the importance of a high-fidelity prototype as the basis of the product spec. I won't repeat all the reasons for that here, but suffice it to say that if your team is remote, you absolutely want to make sure you use the high-fidelity prototype as your main communication mechanism—both for communicating the actual spec and for communicating changes. Written documents are hard enough to get people to read, but if it's written in a language that isn't native, and if the author isn't in the cube down the hall to clarify questions, you're asking for big trouble.

2. When a development team is local, resource conflicts (for example, two different managers give conflicting instructions) are typically caught and resolved quickly. With remote teams, you can expect lots of unpleasant surprises, and literally months can pass before the disconnects are identified. This is usually because the remote developers are forced to make assumptions about who wanted what and how to interpret the various instructions (and,

of course, the assumptions are not always correct). It is critical to have someone local manage all coordination with the remote team. This doesn't mean that all communication must be funneled through this person, but there should be no question to whom the engineering team is accountable. Sometimes this is a project manager and sometimes this is a director or VP of engineering who is based locally (near the product manager).

3. There are many good communication mechanisms available within most businesses today. In addition to e-mail and instant messaging, video conferencing technology is much improved, and VoIP has brought the costs way down for international calls. That said, there still is no substitute for establishing personal face-to-face relationships. At least once a quarter, the product manager should get out and spend some quality face time with the engineering team, meeting with the key architects and managers. These face-to-face visits will improve relationships and communication. Another great communication-building technique is to have an exchange program where key developers come stay with the product manager for a while.

With a talented remote team, and managing the relationship as I've described, you can actually come to enjoy this arrangement. Especially with engineers based in India, the time difference can make it such that every morning when you come in to work, you see progress waiting for you, and you can spend your day (and their night) reviewing, testing, and providing feedback. The result can be extremely fast cycle times.

Note that you can also have your prototyping resources in a remote location, but you'll have to work a little harder on the communication and be more flexible on your hours because of the very quick cycle times (several iterations a day).

Yet another solution to the problems of dealing with a remote development team involves locating the entire product team together in the remote location. I see this trend just starting, and I think it will grow. That said, you don't have to worry about this just yet. It has taken 10 years to develop the engineering and QA capabilities in these remote locations, and it'll likely take another 10 before these locations have skilled product managers and designers as well.

What About Outsourcing?

Just about every company I talk to now is outsourcing to one degree or another. Yet the results are decidedly mixed. I think there are several reasons for the problems that companies are having. Often the problems stem from issues with the product development process, or from language or cultural issues, but more often than not I think the core issue stems from using outsourcing for the wrong reasons.

If you're trying to create inspiring products, for most professional positions, outsourcing should *not* be about cost savings—it should be about assembling the right people for the product. Very often, you'll have to look beyond your immediate geographic vicinity for the best people. This might mean hiring someone that's based in another state, or in another country.

The sad truth about Silicon Valley is that it has become so expensive that many of the people you would like to hire can't afford to live here at the salary you can afford to pay them. Commuting only works to a degree. Eventually the talent supply runs out and you have to look elsewhere for the right team.

Fortunately, there are some terrific sources of outstanding product talent in places such as India, Eastern Europe (especially the Czech Republic, Hungary, Poland, and Slovakia), Northern Europe (especially the Netherlands, Sweden, and Germany), Israel, China, Singapore, Australia, and New Zealand. I know some amazing people in every one of these locations. One of the best teams I ever had the privilege to lead had members spread across Sweden, Silicon Valley, Boston, and India. We were building infrastructure that needed to support more than 20 million users and it's not at all clear we could have succeeded without the talents of the specific individuals involved.

One of my favorite companies is MySQL, which is a company that has embodied this philosophy for years. They are nominally based in Silicon Valley and Sweden, but their product team and even their executives are scattered all over the globe. They are a true virtual organization, and they are able to benefit from some of the very best database and system software talent to be found anywhere. It is not easy for them to manage a completely distributed product team, but I'd argue that very likely they would not have been able to accomplish what they have if they had picked a single location on the globe, and tried to be a typical centralized company.

Just as manufacturing jobs were forced out of Silicon Valley in the '80s, several other types of jobs are moving out today—

especially customer service, QA, and to a somewhat lesser extent, engineering. It's becoming common to have the architects and QA managers located at the company's headquarters, along with product management and design, and then to have the rest of the team at locations either together or dispersed around the globe.

The key is to realize that it's all about the team and the caliber of the individuals it consists of. Many managers don't quite get this, and they are stunned to learn that there can be as much as a 20X difference in productivity among their staff in the same job class. Which will do better—the team that has five very strong people chosen for their proven skills, or the team of 15 that was hired or assembled based on their location? This productivity factor can easily dwarf perceived financial savings. Similarly, it can turn a top engineer living in the very expensive city of Stockholm into a bargain.

There are additional factors that come into play, but it's my firm belief that everything begins with the right product team, and if your product team decides you need to outsource, I hope you do it for the right reasons—to get the right people for your product team and not just because you think you'll save a few bucks.

Engineering Wants To Rewrite!?!

Few words are more dreaded by product managers than being told by engineering: "No more new features! We need to stop and rewrite! Our code base is a mess, it can't keep up with the number of users, it's a house of cards, we can no longer maintain it or keep it running!"

This situation has happened to many companies in the past, and continues to happen today. It happened to eBay in 1999, and the company came far closer to collapsing than most people ever realized. It happened to Friendster a few years ago, opening the door for MySpace to take over social networking. It happened to Netscape during the browser wars with Microsoft, and everyone knows who won. The truth is that most companies never recover.

When a company does get into this situation, the company typically blames engineering. But in my experience, the harsh truth is that it's usually the fault of product management. The reason is that when this comes up, it's usually because for several years, product managers have been pounding the engineering organization to deliver as many features as the engineering team possibly can produce. The result is that at some point, if you neglect the infrastructure, all software will reach the point where it can no longer support the functionality it needs to.

During this rewrite, you're forced to stop forward progress for what the customers see. You might think that the rewrite will take only a few months (more about that below), but invariably it takes far longer, and you are forced to stand by and watch your customers leave you for your competitors, who in the meantime, are continuing to improve their product.

If you haven't yet reached this situation, here's what you need to do to make sure you never do—you need to allocate a percentage of your engineering capacity to what at eBay we called "headroom." Since many of the issues you run into with rapid growth have to do with scale, the idea behind headroom is to avoid slamming into ceilings. You do this by creating room for growth in the user base, growth in transactions, and growth in functionality; essentially, keep the product's infrastructure able to meet the organization's needs.

The deal with engineering goes like this: Product management takes 20% of the team's capacity right off the top and gives this to engineering to spend as they see fit. They might use it to rewrite, re-architect, or re-factor problematic parts of the code base, or to swap out database management systems, improve system performance—whatever they believe is necessary to avoid ever having to come to the team and say, "we need to stop and rewrite."

If you're in really bad shape today, you might need to make this 30% or even more of the resources. However, I get nervous when I find teams that think they can get away with much less than 20%.

If you are currently in this situation, the truth is that your company may not survive. But if you are to have a chance of pulling through, here's what you'll need to do:

Step 1: Do a realistic schedule and timeline for making the necessary changes that engineering identifies. Most of the time in a normal development project, an experienced engineering team will come up with fairly accurate estimates. The exception to this rule is this case of rewrites—here the estimates are often wildly optimistic—largely because few teams have any real experience with true rewrites. You must make informed decisions in this situation, so you'll have to go through every line item on the schedule to make sure that the dates are realistic.

Step 2: If there's any way humanly possible to break up the rewrite into chunks to be done incrementally with user-visible product development continuing on the site, you should absolutely do so. Even though the rewrite might now stretch over two years instead of nine months, if you can find a way to continue to make forward progress on user-visible functionality—even if it's only with 25% to 50% of the capacity—this is incredibly important for the product to stay relevant in the marketplace, particularly in the fast-paced Internet space.

Step 3: Since you'll only have very limited ability to deliver user-visible functionality, you will need to pick the right features, and make sure you define them right.

After eBay's near-death experience, the team made sure they wouldn't put the company at risk again. This meant immediately beginning another rewrite, this time well in advance of issues. In fact, due to their very rapid growth, eBay ended up rewriting a third time, this time translating the entire site into a different programming language and architecture. And they did this massive, multi-million-line rewrite over several years while at the same time managing to deliver record amounts of new functionality and—most importantly—without impacting the user base. It's the most impressive example of rebuilding the engine in mid-flight that I know of.

The best strategy for dealing with this situation is to not get to this point. You need to pay your taxes and remember to dedicate at least 20% to headroom. If you haven't had this discussion with your engineering counterpart, do so today.