do you create products that customers will love?

er you're a product manager or designer on the frontlines, a startup reneur working to identify a winning product strategy, or a product ive trying to ignite innovation, this book shows how the best nies create inspiring and successful products—and how you can too.

At eBay, of all of the leaders in the past decade, Marty had the most significant and lasting impact on how we create products.

Frerk-Malte Feller
Managing Director, eBay Germany

Marty is not only a seasoned expert on all aspects of the often ambiguous discipline of product management, his book also provides nspiration, tools and techniques, and really practical help.

Judy Gibbons
Accel Partners

When it comes to creating inspiring products, Marty Cagan knows his stuff.

Pete Deemer
Former Chief Product Officer, Yahoo! and CEO of GoodAgile

Marty balances key product management principles, great new techniques and examples that bring them all home.

Jim Denney
VP Product Management, TiVo

$29.95
ISBN 978-0-9816904-0-7
52995>
9 780981 690407

INSPIRED HOW TO CREATE PRODUCTS CUSTOMERS LOVE MARTY CAGAN

INSPIRED

HOW TO CREATE PRODUCTS CUSTOMERS LOVE

SVPG PRESS

MARTY CAGAN

Chapter 26:
SUCCEEDING WITH AGILE METHODS

Top 10 List

Many software product teams are either currently experimenting with *Agile* methods, or have recently adopted some form of the methods. While the benefits of *Agile* methods—including *Scrum* and *XP*—are many, most product teams struggle initially as they work to understand how best to apply *Agile* methods which were originally developed for the custom software world to their product software environment.

In this chapter I highlight the keys for succeeding with *Agile* in a *product* software environment.

If you don't yet know what *Agile* methods are, take a look at www.agilemanifesto.org.

Note that this list is meant for product software teams. For custom software, there are some very different considerations.

1. The product manager is the product owner, and he represents the customer. He will need to be extremely involved with the product development team, helping to drive the backlog and especially answer questions as they arise. Some misguided product managers think they get

off easy in an *Agile* environment—they couldn't be more wrong. Some also like to have different people covering the product manager and the product owner role, but this is usually just a symptom of a deeper problem (see the chapter *Product Management vs. Product Marketing*)

2. Using *Agile* is not an excuse for a lack of product planning. As a product manager/owner, you still need to know where you're going, what you're trying to accomplish, and how you'll measure success. That said, in an *Agile* environment, your planning horizon can be somewhat shorter and rolling. You should use the lightweight opportunity assessment instead of a heavy MRD (see the chapter *Opportunity Assessments*).

3. You and your designers should always try and be one or two sprints ahead of your team. This allows you to validate difficult features with sufficient time to improve them. Insist that the designers (interaction designers and visual designers) are front and center in the process, and make sure they don't try to do their design work during the sprint—while the implementation is already underway (see the chapter *Design vs. Implementation*). Make sure, however, that someone from the engineering team is reviewing your ideas and prototypes every step of the way to provide feedback on feasibility, costs, and insights into better solutions.

4. Break the design work into as small and as independent chunks as possible, but not too small—make sure you don't try to design a house one room at a time. But remember the emphasis on coming up with the minimal product possible. Note that, in an *Agile* environment, the designers may need to work faster than they're comfortable with. You'll find that certain designers, and certain design methodologies—such as rapid prototyping—are more compatible with the pace of an *Agile* environment than others.

5. As a product manager/owner, your main responsibility is to come up with valuable and usable prototypes and user stories that your team can build from. Replace heavy PRDs and functional specs with prototypes and user stories. Do prototypes for three reasons: (1) so you can test with real users, (2) to force yourself to think through the issues; and (3) so you have a good way to describe to engineering what you need built during the sprint. Be sure to test prototypes with real users. Try out your ideas and iterate on the prototype until you've got something worth building. You still need to make sure that you don't waste sprint cycles.

6. Let engineering break up the sprints into whatever granularity they prefer. Sometimes the functionality in a prototype can be built in a single sprint, other times it may take several sprints. You will find that having good prototypes will help significantly in estimating the amount of work and time required to build. Remember that the engineering team has considerations in the areas of quality, scalability, and performance, so let them chunk the functionality into sprints as they see fit.

7. Make sure you as product manager/owner and your interaction designer are at every daily status meeting (aka *standup* or daily *scrum*). These morning meetings are the beginning of the communication process, not the end. There will be a constant stream of discussion about the product. Designers should be previewing functionality to the developers and QA. Developers should be showing off completed code to each other, QA, and the designers and product manager. QA and developers should be identifying potential pitfalls during prototyping, and helping the team to make better functionality, design and implementation trade-offs.

8. Don't just launch every sprint—reassemble sprint results in a staging area until you have enough to make a release as defined by the product manager/owner. It's the product

manager's job to ensure that there is sufficient functionality to warrant a release to the user. Remember that in a product environment, constant change can be upsetting to your customers (see the chapter *Gentle Deployment*).

9. At the end of each sprint, make sure you demo the current state of the product, as well as the prototype for the next sprint. Having everyone see what you finished validates the team's hard work, gives the entire company insight into the product, and keeps the evangelism going.

10. Get *Agile* training for your entire team. Hire a consultant to help your product team move to *Agile*, but make sure the consultant has proven experience with product software teams and understands the difference between product software and IT or custom software. If everyone understands the mechanisms around *Agile*, then you can focus on the execution. If people don't understand, you'll get bogged down in the semantics and dogmatic issues.

CAN'T EARLY SPRINTS BE CONSIDERED A PROTOTYPE?

Some *Agile* advocates and practitioners argue that the team should just consider the early sprints as the working prototype. And in fact, for custom software efforts where there really isn't true product management and rarely user experience design, this is the essentially the best you can do. However, for product software organizations, you can and must do better than this, for three reasons:

First, a sprint is typically far too long to wait to try out an idea—an idea which will most likely be wrong. It is much faster to try that idea out with a disposable prototype in days rather than wait months for one or more sprint cycles.

Second, there are typically too many critical things for the engineering team to do to use them for the product discovery process. By taking their time for this prototyping work they are not able to do what they should be doing—building production software.

Third, while *Agile* methods do much to encourage the team to learn and respond quickly, it is still difficult and time consuming for a team to change directions significantly once they have begun down a path, and put long hours into a particular architecture or approach.

CAN AGILE BE USED FOR PRODUCT SOFTWARE?

Agile methods like *Scrum* really do attack some key problems that have plagued software teams for decades. But many product managers and user experience designers—and to a lesser extent QA staff—are initially confused by *Agile* and unsure of their role in these methods. To be clear, these methods absolutely require these roles, but I attribute the confusion to the origin of *Agile* methods. I've found that when I explain the origins, it helps to illuminate the problems that *Agile* was designed to solve, and what challenges remain.

Many are surprised to learn that *Scrum*, the most popular of the *Agile* methods, is now over 20 years old. It was created in 1986 in Japan. (Yet another example of just how long it can take for a new idea to reach the tipping point).

But most importantly, these methods originated in the custom software world.

The custom software world—building special purpose software for specific customers—has long been a brutally difficult type of software. This is partly because customers notoriously don't know what they want, but they have a need so they write a contract with a custom software supplier, or sit down with their internal IT folks, who then work to deliver. When they do deliver, the customer invariably responds that it really wasn't what they had in mind, so the cycle repeats and frustration mounts. But the core need still exists, so this provides job security for countless IT developers, custom software shops, and professional services businesses.

Further, custom software has long been on the short end of the stick when it comes to recruiting and retaining top software talent.

This is partly the case because many top software professionals prefer to work for companies that are in the business of creating software for thousands, if not millions of customers. And partly it's because software professionals get paid more working for product software companies where the product team is responsible for coming up with software products that please many people, or they don't make money. So these companies know they must hire the talent necessary to create winning products, and they pay accordingly. But to put this in perspective, only a relatively small percentage of software people actually work on commercial product software—most work on custom software.

In the custom software model, since the customer believes he knows what he needs, you'll rarely find the role of the product manager. Likewise, you'll almost never find user experience designers. The reasons for this are more complex, and involve a degree of ignorance (relatively few in the custom software world realize what user experience designers do and why they're needed), and cost sensitivity (cut costs by letting the developers design). But to be fair—due to the shortage of user experience designers in our industry—the few available are immediately grabbed by the product companies that realize how critical they are, so custom software teams can rarely find designers even if their leaders realize they need them. Similarly, QA as a discipline is rarely found in custom software projects—again, the developers are typically expected to do the required testing.

Another crucial element in understanding the custom software world is that the vast majority of custom software projects are relatively small and done to support the internal operations of a company—applications such as HR, billing, and manufacturing—where the limited number of users means that issues such as scalability and performance are usually less critical.

Historically the custom software world used the Waterfall process because the various stakeholders needed a way to monitor progress during the long process of creating these contract applications. In fact, the Waterfall methods originated here as well.

In the product software world, where the software must sell on its own merits, we introduced the roles of product managers to represent the needs of a wide range of customers, user experience designers to create effective user experiences, and QA testers to ensure the software worked as advertised in the range of customer environments.

But in the custom software world, the same fundamental issues of coming up with something that satisfied the customer continued.

For these teams, especially, the *Agile* methods represent significant improvements. They improve communication between the customer and the engineers. They significantly reduce the risk by building smaller, more frequent iterations so that the customer can learn whether he really likes something or not much sooner—rather than waiting for the end of a long process. They help introduce some modern software testing concepts, and they help relieve the team from spending countless hours preparing documents that are rarely read—and quickly obsolete.

In general, these are great benefits for product software teams as well, but I always explain that a few adjustments are required. I've written earlier about these topics—such as how to insert user experience design into the process, and how to manage releases and deployments—but another area that has struggled is architectural design.

Agile methods encourage engineers to not get attached to their implementation, believing that things can be re-factored or re-architected relatively quickly and easily. This is true for the vast majority of custom software, but for many product software systems, such as large-scale consumer Internet services—which must support hundreds of thousands if not millions of users—this approach can be naïve.

So it shouldn't be a big surprise that the main issues many product software teams encounter with *Agile* methods stem from their custom software origin. Many *Agile* books, articles, and training classes still don't mention product managers—or any form of user experience designers (interaction designers and visual designers)—because they aren't meant for product software teams.

My suggestion to teams moving to *Agile* is to make sure the firm you hire to help your organization transition to *Agile* actually understands the differences that product software demands. Most don't, but some do.