PM at Microsoft

By Steven Sinofsky
Published December 16, 2005

While at Stanford this week I was asked by a number of PM (program manager) candidates to talk about the PM role at Microsoft. The PM role is unique to Microsoft and was actually created in response to developing software that is more usable and at the same time pushes the state of the art of technology. So when we talk about PM at Microsoft, we're talking from a perspective of creating and evolving the role over the lifetime of the PC industry.

I have been both a PM and an SDE (software design engineer) during my career at Microsoft. When I was recruited I started off as an SDE candidate and then I learned about PM during the course of my interviews and I thought "COOL!" that has to be a job for me--after all it sure sounds like an incredibly cool role, since it has the title "manager" in it and if you read/hear the description it sure sounds like you're running the show. The job title "program manager" is a bit of a misnomer, since program managers do not program nor do they manage--go figure. I went with SDE because I wanted to write code in my job. After being an SDE for 4 years I moved to program management. I think that was a good move for me because while I like to think I was competent at development, I was probably never going to hit it out of the park with my ability to write code, but I think what I wasn't able to do in writing code I made up for with the skills required of a program manager 🙂

What follows is a description of program management from a PM perspective -- that means through the lens of a PM. It is also an analytical description based on my experience. I'm not trying to hype the job or make it seem too over the top. In fact, I've recently talked with an number of candidates who have been told of PM roles at other companies and my feeling is that they are not being told the whole story. So with my blog I try to be a bit more complete and not "sell". This might make it seem like there's no room for change or to impact the definition of the role, but that is not the case. What I'm describing is in constant evolution and it is the members of the team that are constantly evolving the role. PM is a role that has a lot of responsibility but you also define it--working in partnership with expert designers, clever developers, super smart testers, etc. you all work together to define the product AND the responsibilities each of you have. **PM is one equal part of a whole system.**

Program managers got started at Microsoft while developing Excel for the Macintosh. The challenge the company saw was that we were stretched thin trying to push the state of the art in technology (in this case develop for a brand new OS using brand new Graphical User Interface concepts) and so the developers were very busy just trying to make things like printing, display, graphics, etc. work as well as trying to develop new algorithms for spreadsheets and modeling. That meant that the place where we were not focusing enough attention was on the usability or the scenarios for how people would use the software. In a very classic sense the high order bit of our development was the code--this is pretty classic in most organizations today where you really only see development and test (sometimes called QA) and to the degree that there is input from users/customers this is usually done by the demand generation or marketing team (called product managers in Silicon Valley). So at Microsoft a new role was created in Program Management with the explicit goal of partnering with development and working through the entire product cycle as the advocate for end-users and customers.

The challenge of trying to write code using the latest technologies and bringing ever more complex scenarios to end-users in a simple way continues today as we build web sites (OfficeOnline), server products like SharePoint, and entirely new products like OneNote, as well as build whole new user experiences to replace the aging menus/toolbar paradigm.

Up front I would say that the PM role at Microsoft is both unique and one that has unlimited potential -- it is PMs that can bring together a team and rally them around a new idea and bring it to market (like OneNote, InfoPath). It is PMs that can tackle a business problem and work with marketing and sales to really solve a big customer issue with unique and innovative software (like SharePoint Portal Server). It is PMs that can take a technology like XML or graphics and turn it into an end-user feature benefitting 400 million people (Office Open XML format or the new graphics functionality in Office12). I could go on and paint a very emotional picture, but let's spend some time digging into the more analytical aspects of the job.

Where developers were focused on code, architecture, performance, and engineering, the PM would focus on the big picture of "what are we trying to do" and on the details of the user experience, the feature set, the way the product will get used. In fact the job has matured significantly and it is almost impossible to document a complete list of the responsibilities of program management. One way to do that is to list the sections of a specification for features in Office that a PM would complete before the code gets written which includes nailing the scenario you are solving for, the worldwide implications (will your work make sense to customers in China, India, Estonia), how will developer customers extend the product (object models, programmability), is the product secure and is privacy maintained, what are the other features your work interacts with, will the feature be responsive and performant, and more. These are critical parts of the design of any product, and when you will have 400 million potential customers thinking these through before you start the work is critical.

As an aside a lot has been said lately about "agile development". A key benefit of program management is that we are far more agile because we have program management. That can be counter-intuitive (even for many developers at Microsoft who might be waiting for their PM to iron out the spec). But the idea that you can just start writing code without having a clear view of the details and specification is a recipe for a poorly architected features. A great PM knows when the details are thought through enough to begin and a great developer knows when they can start coding even without the details for sure. But like building a house--you don't start without the plans. While software is "soft" the cost of remodeling or rebuilding is no different than if you decide the walls are in the wrong place or you forgot to leave room for the HVAC system--sometimes because we don't have material costs in software we think that "rewriting" is perfectly ok. That generally works super well in two cases. First, if the program is small then of course you can rewrite it. In the commercial software world most software projects are not the work of one or two people--in fact one way to think of it is that if a project is only the work of one or two people (or is only a 100KLOC) then the economic value (and thus the impact) of that product is probably pretty finite (at the very least a competitor is likely to catch up very quickly). And second, rewriting works well when you are looking backwards and re-doing what has been done (like cloning an existing product, implementing something for the n-th time). When you're doing innovative work, the time spent on design and analysis pays off handsomely in the ability to develop a complete program and to have something of lasting economic value. AND when you have a process that embraces design and up front thinking you also find your development much more agile because when you do get the code in the hands of customers--after 2 months or 2 years--there is time to respond and react with changes because you are not overwhelmed with the basics.

The last point is worth re-emphasizing. There's a school of thought that just getting your software out in beta and then "reacting" to the feedback is the best way to get the product done. Again, this tends to work for products that already have a definition you are chasing of if you're interested in incremental feedback. So if you're building an email experience and release it in beta, your customers/users can help you to prioritize which features in the universe of email to add next assuming you did not have all of them up front. But if you're doing something new and innovative or more importantly if you want more than incremental feedback then you need to have much more sophisticated mechanisms than just beta feedback from power users. Most importantly, the role of PM is to represent all customers, not just the ones who do beta tests or the ones who take the time to send in feedback or use early products. A key skill of program management is to have empathy with a broad range of customers. Often when you are just getting started you will see how easy it is to over-emphasize early power user feedback or anecdotes over broad based feedback. Don't get me wrong, power users are great but they are just one type of user. A great advocate for the "little guy" is Walt Mossberg and he really does point out when you're missing the mark on a product and too focused on "techies" as he calls them. The bottom line is that most people are not techies, but most beta customers and most early adopters are so you as a PM have to do the leg work to validate your work with a broader audience.

Before talking about what PM does specifically it is important to look at PM in the context of the relationships with the others that contribute to building products. A PM serves as the coordinating point where all the disciplines come together--this is why you can think of the PM as being the hub. What is critical about the Microsoft PM role is that all the different people serve as a "virtual team" building the product--the PM does not have to go pull them together but rather has to help get everyone aligned. The resources for a project are dedicated to the project--these resources include development, test, product planning, usability, and product design. Each role brings a unique perspective a unique set of skills to the product development process. While you can often be someone who has an affinity for another discipline, rarely can any one person bring all of these skills and experiences. And I can promise that any project of significance really needs the specialized skills to do a world class job and build a sustainable and uniquely innovative product. I should probably write a blog like this on each role--maybe after the holidays!

- **Development** -- developers write the code. They are in charge of the code's architecture, the performance, the reliability, the security, and getting the functionality into the product. Of course development is at the core of what we do as a software company, but it cannot stand on its own.
- **Test** -- software design engineers in test insure the quality of the product but more importantly that the product ultimately meets the needs of the customer we set out to meet. SDETs will review all specifications as first class members of the team and use their perspective and experience in working with PM and SDEs to insure that the product is not going to be fragile and that it will be possible to insure a successful implementation.
- **Usability** -- usability engineers validate the designs of our software and incorporate the statistical understanding of customers into the process. Microsoft was an early pioneer in integrating usability in the product development process (though some might say how in the world we released the Hot Dog Stand theme if we had usability). Many usability team members have PhDs in HCI or related research areas such as anthropology, and many are undergraduates with a HCI or psychology background. These team members design mechanisms to test and validate designs and design assumptions.

- **Product Design** -- Design work in partnership with PM to develop the interaction model for our products and also own the overall product look and feel. While many companies use design for "graphical design" (which we do) our designers are expert in computer interaction--many have studied at programs like Delft's or CMU. When you look at the new user interface in Office12 what you see is a strong collaboration between the PM experts and the Design and Usability experts.
- **Product Planning** -- our planners are experts in understanding the market place and understanding what our customers need from software. Planners own research and communicating that research to the product team PMs and to the executives deciding the overall goals of a release. Planners are assigned to broad technology and business areas (collaboration, business intelligence) where they become expert in all the products and solutions on the market and how Microsoft can offer customers improvements over what is in the market. Many planners have business background and have worked in marketing before.

Many companies will "sell" you on being able to do many of these different things from one job. This is just not a reality that exists and I always feel a bit bad for folks who believe this. There are two times I hear this a bunch. First is at startups you hear "come join us from college and you can own all of this". Of course at a startup the real experience is that you are the college hire, which means you will do the grunt work while the founders and the venture people do all the strategic work--so you might find yourself setting up the build machines rather than interacting with customers. Second, I hear this a lot when companies are selling against Microsoft and point out that "at our company we do not specialize and everyone does everything". This is another "well in reality..." situation, since of course even when I have seen companies that claim to do the specifications or customer research and up front planning they do that work from Product Management, and those people are just as specialized, they just report to the marketing team. And we know what that means, which is when push comes to shove the marketing team will need to use every hand to get out there and generate the business and sell--so even if there is a single group that does the work, those roles are specialized, and rarely dedicated specifically to the role. These are my experiences and of course your specific situation might be different. I've just seen these patterns repeat themselves over many years of hiring students and having them come back to Microsoft after a short experience with something like that.

It is worth talking about the size of the PM team for a bit. In Office our PM teams are small (our dev teams are usually about 20-30 developers as talked about previously). The entire user interface for Office12 was developed by a team of about 12 program managers--imagine that 12 program managers did all the work to totally define the new interaction model for 400M customers of Office. But along with the fact that the team is small is the fact that a team like this is also one where even the newest members of the team receive significant mentorship and training from a very senior member of Microsoft (you can watch this video to meet the head of the user interface PM team). So you have the biggest impact you could have in designing software while receiving a very high level of management attention. And then of course each PM has developers they are responsible to equip with features and specs -- the whole user interface development team was less than 30 people (so about 2 devs for each PM). A critical part about PM (also one that is unique for Microsoft) is that as a PM you have dedicated developers for your feature area--your job is to get the most out of those developers in terms of bang for the buck by having great feature ideas and great specs to get them done.

So what does a program manager do? Well there are three phases to program management: learn, convince, spec, refine. These roughly mirror the product development timeline written about previously. Do keep in mind that the timeline is not fixed--rather the phases of the timeline are. So if we're doing a 12 month project then the time is divided accordingly, same as if the project is 24 or 30 months.

Learn --> Output of learning is a prototype

During the learn phase, program managers spend their time learning about customer problems and learning about the products and technologies out there that are relevant to these problems. There is a lot of work with planning to understand competitive products or new products in the marketplace. As you can imagine, customers have huge challenges in getting their computing systems to do what they need. So often we will spend days at a customer's place of business learning from them and watching them trying to get their work done. A great example of this is the work we did to understand how customers manage documents in an organization. It is easy to see that organizations like law firms or pharmaceuticals are heavily dependent on the flow of documents in an organization (contracts, drug approval and research) and so the systems are both mission critical and elaborate. Companies really want to automate more and to make things more usable and reliable. The work of program management was to deeply understand how to model the flow of documents in an organization--spending time at big drug companies, small startups, big law firms, local law firms, Public Relations agencies that produce press releases, or government offices that produce legislation to name a few. Out of this work PM and Planning developed a conceptual model called the "document lifecycle" (DLC). This helped us to frame the way that customers would like features to work. So for this work the PM needs to be really good at working directly with customers, learning from them, listening, being open minded to different ways of working, etc. When you're hired from college you will participate in this work for your area.

At the same time there are deep technical problems to solve. We need to solve the control and access to information (drug test results and contracts need a high degree of security, yet they need to be collaboratively authored). PMs spent a lot of time working with the Windows platform team who develop platform technologies like our Rights Management Server or the Windows Workflow Foundation (WinWF). These technologies are critical to enabling a robust and extensible implementation of the DLC. So in this phase of learning, the PM has to become well-versed in new platform technologies or in how to apply existing technologies. Often this is where some people say "it would be easier to roll our own" -- which of course in the immediate term it is (just build your own linked list and event source/sink) but what you miss out on is the expertise and leverage that comes from a deep platform technology. For example, by learning the WinWF and being an ISV of that technology we can take advantage of advances in workflow, integration with Visual Studio, and a very robust and scalable server without us doing any work--just like when developers reuse the process model of Windows, or the client side drawing code of GDI.

With this information at hand the role of the PM is to synthesize this learning into a series of prototypes. These prototypes are of varying fidelity. This is where the analogy to architecture holds--sometimes you do a drawing, sometimes you do a high fidelity diagram, and sometimes you build a model. In software we sometimes build static screen shots, we sometimes prototype in tools like VB.NET, sometimes we prototype in a series of static bitmaps that illustrate a scenario. For our

new user experience in Office12 we went through a series of high fidelity prototypes development first in PowerPoint (you'd be amazed at the depth of interaction you can do) and then in more high-end design oriented tools. By the time we were ready to exit the learning phase, we had a full mockup of the user interface (which I have shown at my campus tech talks this year).

The output of this phase is a prototype upon which we will author specifications.

Convince --> Output of convince phase is a plan and goals

Once you've learned a lot you have to go out and convince your peers, your manager, and the dev team that your ideas are worth pursuing. This is where being a solid communicator and someone who is able to bring together customer needs, technologies, and scenarios really comes together.

As an aside, a lot of companies will tell you that you can come and pursue your ideas. That is probably not a good plan -- that is a recipe for chaos. But more importantly, if you can do whatever you want there is a good chance someone else in the company has the right to come in and tell you to change. If there is this level of empowerment it means the management team is empowered as well 🙂 The ability to just start at a company and pursue your own agenda is much more of a lore than any reality--all companies have a strategy and a business framework that the work needs to fit within. At the very least, eventually shareholders will want a return on your R&D investment.

So at Microsoft the convince phase is really where your entrepreneurial skills come to play. While you will always have work to do, during this phase you are working to expand your vision and get as many lined up behind your area as you can handle. Your ability to convince people of your ideas is a lot like trying to get funding from venture capital folks. That is why if you have a great conviction of your ideas and a lot of energy you probably have the makings of a great PM.

The best part about this is that the teams you work with are all working in unison on the vision and everyone is on board trying to make sure the ideas come together super well. In particular your mentor is there to work with you. But ultimately, the ideas will succeed based on your own initiative and ability to convince the team of the chances for success and the high priority nature of the work.

The output of this phase is the set of objectives for the project area. What follows is developing things at the next level of detail.

Spec --> Output of spec phase are a series of written specifications

The first thing people always think of is "specs are for bureaucrats". That drives me a bit crazy. I know as Americans we have a tendency to use new products without reading the instructions, but it is lunacy to develop a new product without first writing down the goals. The mere process of writing is thinking, and the thinking will always push you to uncover more issues sooner before it is way too expensive to change things. For all the talk about agile development, few ever talk about

specifications as being the most important step in agility. It is way easier to change a bitmap or do some editing of English in Word than it is to move around and rearchitect code.

Yet at the same time we do not sell our specifications, we only sell our code. So while we work to be very hardcore about having up front planning we do not develop our specifications to the point that they are the full documentation of the product. It is too much work and not enough of a pay off. So if you make a late breaking design change we might document the change in the change log but we don't go back and redo all the words in the spec. Another fun one for reading specs from a long time ago is that the actual feature name used in the product is never what we named it during development--the Office Assistant was famously named TFC during development. The "C" stood for clown. I will let your active imagination figure out what the TF stood for.

So in writing a specification, the PM that worked on the learn phase now has to turn that learning into an experience that customers will have. This is where the entire series of features, user interactions, boundary conditions, localization, ISV opportunity, and all the details are filled in. A specification is how a developer will estimate the amount of time it will take to write the code. So if your spec omits a lot of details and developers need to guess, then there is a good chance you will end up not being able to realize all of your dreams because too many things needed to get filled in a the last minute, plus your developers will not be very happy working with you. So doing a good job on the spec is important.

A great program manager will take their whole feature area and divide it up into specifications that match the developers or break the feature up into workable "chunks". On average a PM writes about 8-10 specs for their area, and each one can be 30-50 pages depending on how visual (how many pictures). Some specs are significantly longer and some PMs choose to write a larger number of smaller specs.

Specs are not just for developers. But the usability, product designers, and testers are all contributors to and refiners of the specifications. In fact while the output of the Spec phase is the document, the final output is an "inspected specification". If you've ever gone through a code review with a professor or mentor (as an intern) a spec inspection is like a code review. During this time testers might push on you about boundary conditions or compatibility with existing products. Product designers might work with you to improve the way the spec describes the overall interaction. Usability might research the instrumentation from in-market products and use that to refine the priorities for the feature. In fact the role of data is critical to Office PMs--if you run a web site you have click streams and logs, and Office through the Office Customer Experience Program has exactly the same--usage information for millions of volunteers (anonymous and private of course) and that educates us immensely on how to design features (Jensen's blog describes this as well). So the spec, while owned by PM, is a work product of many contributors.

It is worth noting that many times along with a spec is a more detailed prototype. This is particularly true for heavily interactive features. In this case product design and PM work together to develop detailed prototypes of the work (often these will be tested in the labs with customers as well).

When the spec is complete the core part of development begins. PM then transitions to refining the product.

Refine --> Output is the product

During the development of the product, PMs are constantly refining the details, working with development and test to determine what needs more clarity and what is working better than expected (that can happen) and what is going less well (that happens more). PMs are on call to answer questions and "unblock" development.

More importantly as the real code becomes available, PM is anxious to try it out and make sure it is meeting their own design expectations. Once the real code creeps along we will bring that into our usability labs to further understand how the product works. A famous story of this phase of developing the PM role was that the first PMs were trying to improve a feature under development and ran some usability tests. The feature bombed. The PMs tried to explain this to the developers, but the developers insisted that the PM just picked "dumb users" because the feature was great. So after a few rounds of this the PM dragged the Dev to the tests to watch 10 out of 10 users fail to use the feature. The developer finally relented and the feature was fixed. Today, developers can watch tests via streams or go downstairs to our usability labs and watch the tests in person. Almost all developers I know are keenly interested in how their features are doing (and how the PMs are doing designing those features!)

Another aspect of refinement is working with customers who use the early code. While it is cool to throw a product out to beta and get some instrumentation and maybe get some qualitative input via mail, the only true way to understand how the product is doing is by deep engagement with real world users. We do this this through a number of mechanisms that gradually expand the number of customers involved.

Early in the process we meet with a very small number (10-20) customers who spend days with us here on campus and learn about the product. We walk them through all the details and solicit their feedback and input. We did this for Office12 last summer and it was critical to our ability to get to Beta1 with a product that reflects real world usage and learning. As a PM you will be responsible for getting together with these customers and learning from them. We often follow up on site.

Another group we learn from that is a bit larger are our MVPs -- the most valued professionals. These folks are the power users of Office. Our PMs all got involved with the MVP summit on campus and again worked with the MVPs in small groups to get their feedback and expertise into Office12. The MVPs know more about Office than any other customers on earth--they are a wealth of information.

We're currently in the phase of learning from a broad set of beta1 customers. We are doing this through newsgroups and through direct communication.

You might also notice that many of our PMs are blogging (see the links to the right). This is new for us (well for everyone) and also a great source of information and a great way that PMs are connecting with the web community.

Of course our senior program managers are also responsible for working with the press and product management. So a lot of time is spent on communicating Office12. There are a lot of reporters interested in Office and a lot of information to get out there.

All of this Refine work feeds back into the developers where we prioritize and make sure that the very best product is built. Of course that doesn't end with RTM since we're always refining and listening to customers (even though we're not "Web 2.0"). As I mentioned previously, we make over

100 changes every month to Office based on customer input -- so the product is always improving, and we're always learning!

PM Attributes

So those are the phases of program management and what you do as a PM in Office. I would say that there are many unique elements to the role of PM in Office that are not emulated by other companies who have tried to create this role. A good book that describes the uniqueness of PM at Microsoft is Michael Cussumano's book "Microsoft Secrets" or his new book, "The Business of Software". If you're considering a PM role at Microsoft (or Office), from my perspective a couple of things you will get:

- A small team dedicated to solving customer problems and bringing the best technologies to the table
- Access to dedicated developers who are there to implement your ideas if you can live up to creating a great idea and making sure the details are thought through
- A very strong mentor who as part of small team will be there for you on a daily basis

If I had to think of the qualities that make a great PM I might list a few, but your recruiter and the interviews will help out a lot so don't let these discourage you from applying:

- Strong interest in what computing can do -- a great interest in technology and how to apply that to problems people have in life and work
- Great communication skills -- you do a lot of writing, presenting, and convincing
- Strong views on what is right, but very open to new ideas -- the best PMs are not necessarily those with the best ideas, but those that insure the best ideas get done
- Selfless -- PM is about making sure the best work gets done by the team, not that your ideas get done.
- Empathy -- As a PM you are the voice of the customer so you have to really understand their point of view and context
- Entrepreneur -- as a PM you need to get out there and convince others of your ideas, so being able to is a good skill

PM is unique to Microsoft and I think it is fair to say this is a role that is often copied but never duplicated.